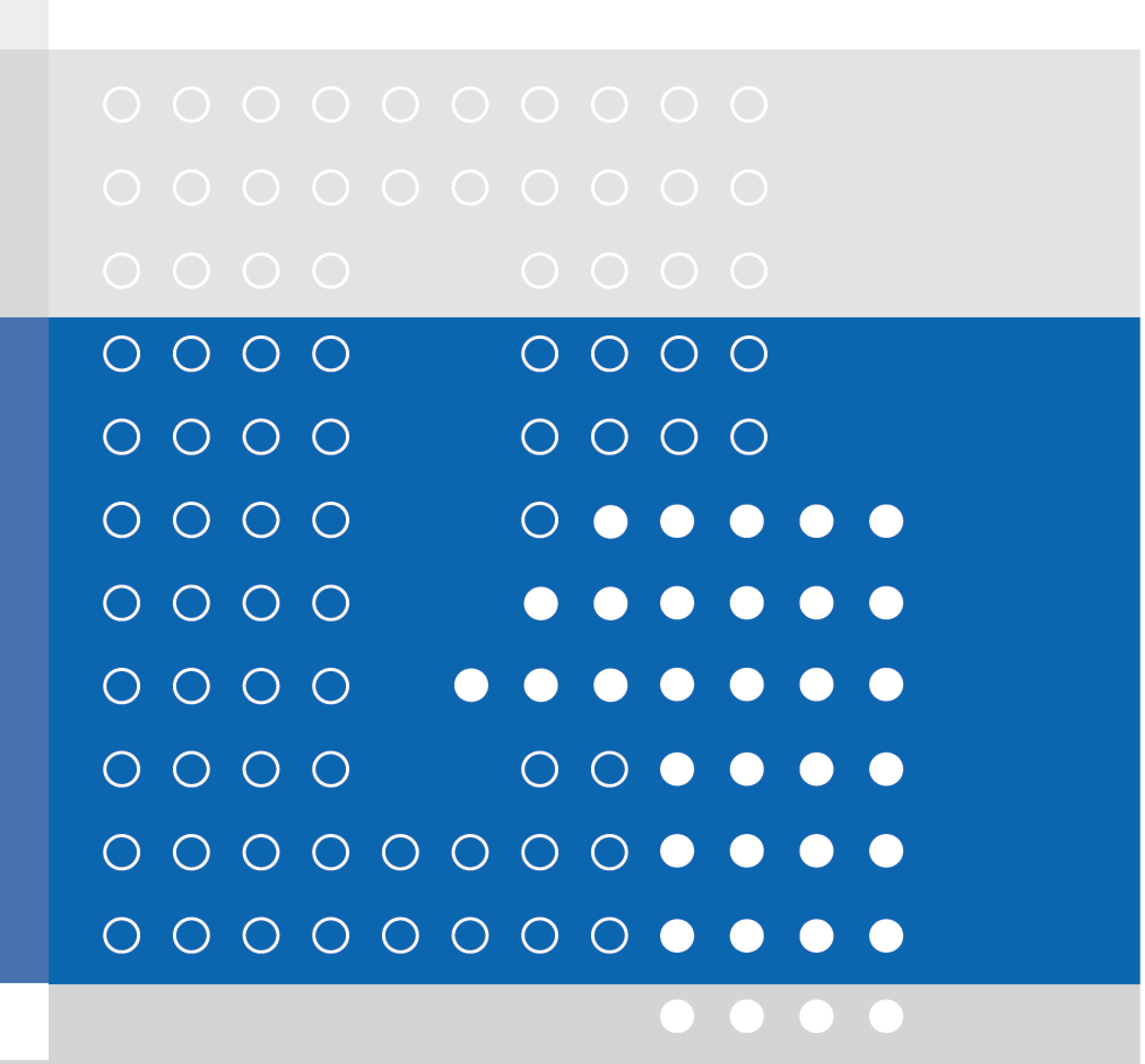


普通高等教育“十一五”国家级规划教材　计算机系列教材

软件测试理论与技术



李千目　主编

清华大学出版社

计算机系列教材

软件测试理论与技术

李千目　主编

清华大学出版社
北　京

内 容 简 介

全书共 17 章，第 1～第 8 章围绕软件测试基础理论进行阐述，从第 9 章开始引入性能测试，在后续章节中描述了性能测试原理、应用领域、团队建设、测试工具、需求分析、Web 性能测试等内容，第 9、第 10、第 11 章分别阐述了软件性能测试的基础理论、应用领域和团队建设，第 12 章针对性能测试工具原理进行介绍，第 13、第 14、第 15、第 16 章以性能测试流程为主线，对性能测试需求分析、测试脚本编写、测试场景设计与执行以及测试结果分析的技术要点分别做出了详细介绍，第 17 章针对性地介绍了 Web 前端性能。

软件测试是发现软件缺陷最有效的手段，而完备的性能测试是最关键的。通过负载测试、压力测试、配置测试、并发测试、可靠性测试以及失效恢复测试等一系列方法，性能测试在能力验证、能力规划、性能调优和缺陷发现等领域大显身手。从软件行业本身的需求而言，高级性能测试人员的市场需求十分巨大，性能测试行业是十分有发展前景的。

本书以介绍软件测试基本理论为引子，围绕性能测试原理、方法与实践展开探讨。

本书可以作为高等学校计算机科学与技术、软件工程、网络工程和通信工程等专业的本科生或研究生教材，也可以作为相关领域工程技术人员的参考书。

图书在版编目（CIP）数据

软件测试理论与技术/李千目主编. --北京：清华大学出版社，2015
计算机系列教材
ISBN 978-7-302-39982-7

Ⅰ. ①软… Ⅱ. ①李… Ⅲ. ①软件－测试－教材 Ⅳ. ①TP311.5

中国版本图书馆 CIP 数据核字(2015)第 086509 号

责任编辑：谢　琛　薛　阳
封面设计：常雪影
责任校对：李建庄
责任印制：沈　露

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 装 者：三河市少明印务有限公司
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**15　　**字　　数：**376 千字
版　　次：2015 年 9 月第 1 版　　**印　　次：**2015 年 9 月第1 次印刷
印　　数：1～2000
定　　价：30.00 元

产品编号：061669-01

前 言

“性能测试”是什么？很多人即便没有从事过性能测试工作，也大都听说过这个名词。然而从企业实践的角度，究竟什么是软件性能？软件测试中的性能测试应该如何开展？对于性能进行测试要通过哪些指标来着手？

当今社会，电子商务已经不是什么新鲜名词。大家都有网购的经历，也都通过电子银行进行账户管理。然而，涉及金钱，就不能不对安全性多加关注。举个例子而言，客户在银行存款，存款额度不能无缘无故减少，并且交易记录可查，这是最低标准。不能说因为某次交易量过大，发生系统崩溃，所有数据永久性丢失，这是极为严重的灾难。性能测试就做这些，它的关注点已经不再是软件是否实现什么样的功能这样的层面，它关注的是客户的交易感受是否迅速、在突然增加巨大交易量的前提下系统是不是能够从容应对等问题，通过本书的学习，读者会知道这涉及性能测试中十分重要的概念——负载测试及压力测试。这里只是举出了这样一个简单的例子，实际性能测试的内容要多得多。

对应用系统本身性能的关注和软件性能的复杂性催生了专门的“性能测试工程师”这一职位。根据一些业内人士的统计，“性能测试工程师”无论从升职空间，还是薪资标准上都不亚于开发人员。从国内从业人员的数量和质量指标来看，国内的性能测试人员缺口还是比较大的，尤其是能够统筹规划、承担大型系统性能测试工作的人员更加稀少。

软件性能测试是十分复杂的过程，它涉及的因素十分庞杂，包括网络环境、数据库服务器、应用服务器、业务逻辑的实现方式、系统采用的架构、代码优化速度、使用者的使用方式都会对软件的性能表现造成影响，在这种局面下，性能测试的开展是十分不易的。

本书面对这种挑战，将从性能测试的各个方面展示性能测试的原理，实施过程等内容，力求全面而又系统地阐述企业性能测试的过程，使读者具备性能测试实践的能力。

1. 本书特色

本书以软件测试的基本概念为出发点，讲述软件测试的基本原理。在使读者具备测试理论的知识储备之后，引入性能测试。从性能测试基础理论，到性能测试的工具、应用以及团队建设等方面，系统而又详实地介绍性能测试。本书的行文得到了测试专家 Kern Zhang 的许多宝贵指导意见，相信是性能测试学习者的一个不错的选择。

2. 本书组织

本书第 1～第 8 章围绕软件测试基础理论进行阐述，从第 9 章开始引入性能测试，在后续章节中描述了性能测试原理、应用领域、团队建设、测试工具、需求分析、Web 性能测试等方方面面的内容。

本书由李千目主编，陶传奇、刘晓迁、陆建峰、许春根任副主编。何光明、王珊珊、卢振侠、石雅琴、杨橙、陈莉萍、陈凤、曹冬梅等也参与了本书的部分编写工作。

编　者

2015 年 6 月

目 录

第 1 章　软件测试的基本概念

1.1　软件质量

1.1.1　软件质量的概念

1979 年，Fisher 和 Light 将软件质量定义为：表征计算机系统卓越程度的所有属性的集合。1982 年，Fisher 和 Baker 将软件质量定义为：软件产品满足明确需求的一组属性的集合。20 世纪 90 年代，Norman、Robin 等将软件质量定义为：表征软件产品满足明确和隐含需求的能力的特性或特征的集合。

1994 年，国际标准化组织公布的国际标准 ISO 8042 综合将软件质量定义为：反应实体满足明确的和隐含的需求的能力的特性总和。

综上所述，软件质量是产品、组织和体系或过程的一组固有特性，反映它们满足顾客和其他相关方面要求的程度，如 CMU SEI 的 Watts Humphrey 指出："软件产品必须提供用户所需的功能，如果做不到这一点，什么产品都没有意义。其次，这个产品能够正常工作。如果产品中有很多缺陷，不能正常工作，那么不管这种产品性能如何，用户也不会使用它。"而 Peter Denning 强调："越是关注客户的满意度，软件就越有可能达到质量要求。程序的正确性固然重要，但不足以体现软件的价值。"

GB/T 11457—2006《软件工程术语》中定义软件质量为：

(1) 软件产品中能满足给定需要的性质和特性的总体。

(2) 软件具有所期望的各种属性的组合程度。

(3) 顾客和用户觉得软件满足其综合期望的程度。

(4) 确定软件在使用中将满足顾客预期要求的程度。

1.1.2　软件质量的属性

1. 正确性

正确性(Correctness)是指系统满足规格说明和用户目标的程度，即在预定环境下能正确地完成预期功能的程度。

2. 健壮性/鲁棒性

健壮性(Robustness)是指在异常情况下(如硬件发生故障、输入的数据无效或操作错误等)，软件能够正常运行的能力。健壮性有两层含义：一是容错能力，二是恢复能力。

容错是指发生异常情况时系统不出错误的能力，对于应用于航空航天、武器、金融等

领域的这类高风险系统，容错设计非常重要。

恢复则是指软件发生错误后（不论死活）重新运行时，能否恢复到没有发生错误前的状态的能力。

3. 可靠性

可靠性（Reliability）是指软件系统在一定的时间内无故障运行的能力。

可靠性是一个与时间相关的属性，指的是在一定的环境下，在一定的时间段内，程序不出现故障的概率，通常用平均无故障时间（Mean-Time To Fault，MTTF）来衡量。

4. 性能

性能（Performance）是指软件及时提供相应服务的能力，具体包括速度、吞吐量和持续高速性三方面的要求：

速度往往通过平均响应时间来量度；

吞吐量通过单位时间处理的交易数来量度；

持续高速性是指保持高度处理速度的能力。

5. 安全性

安全性（Security）是指软件同时兼顾向合法用户提供服务，以及阻止非授权使用软件及资源的能力。安全性既属于技术问题又属于管理问题。

6. 易用性

易用性（Usability）是指用户使用软件的容易程度。软件的易用性要让用户来评价。

7. 可用性

可用性（Availability）指的是产品对用户来说有效、易学、高效、好记、少错和令人满意的程度，即用户能否用软件完成他的任务，效率如何，主观感受怎样。

8. 互操作性

互操作性（Interoperability）是指本软件与其他系统交换数据和相互调用服务用以协同运作的难易程度。

9. 易理解性

易理解性（Understandability）是指理解和使用系统的难易程度。

10. 可扩展性/灵活性/适应性/可伸缩性

可扩展性（Extensibility）反映软件适应“变化”的能力，调整、修改或改进正在运行的软件系统以适应新需求、变化了的需求的难易程度。

11. 可重用性

可重用性(Resuability)反映了重用软件或其中一部分的难易程度。

12. 可测试性

可测试性(Testability)指对软件测试以证明其满足需求规约的难易程度。

13. 可维护性

可维护性(Maintainability)指为修改 Bug、增加功能、提高质量而诊断并修改软件的难易程度。

14. 可移植性

可移植性(Portability)是指软件不经修改或稍加修改就可以运行于不同软硬件环境的难易程度,主要体现为代码的可移植性。

1.1.3 软件质量的模型

1. Bohm 质量模型

Bohm 质量模型是 1976 年由 Bohm 等提出的分层方案,其将软件的质量特性定义成分层模型,如图 1-1 所示。

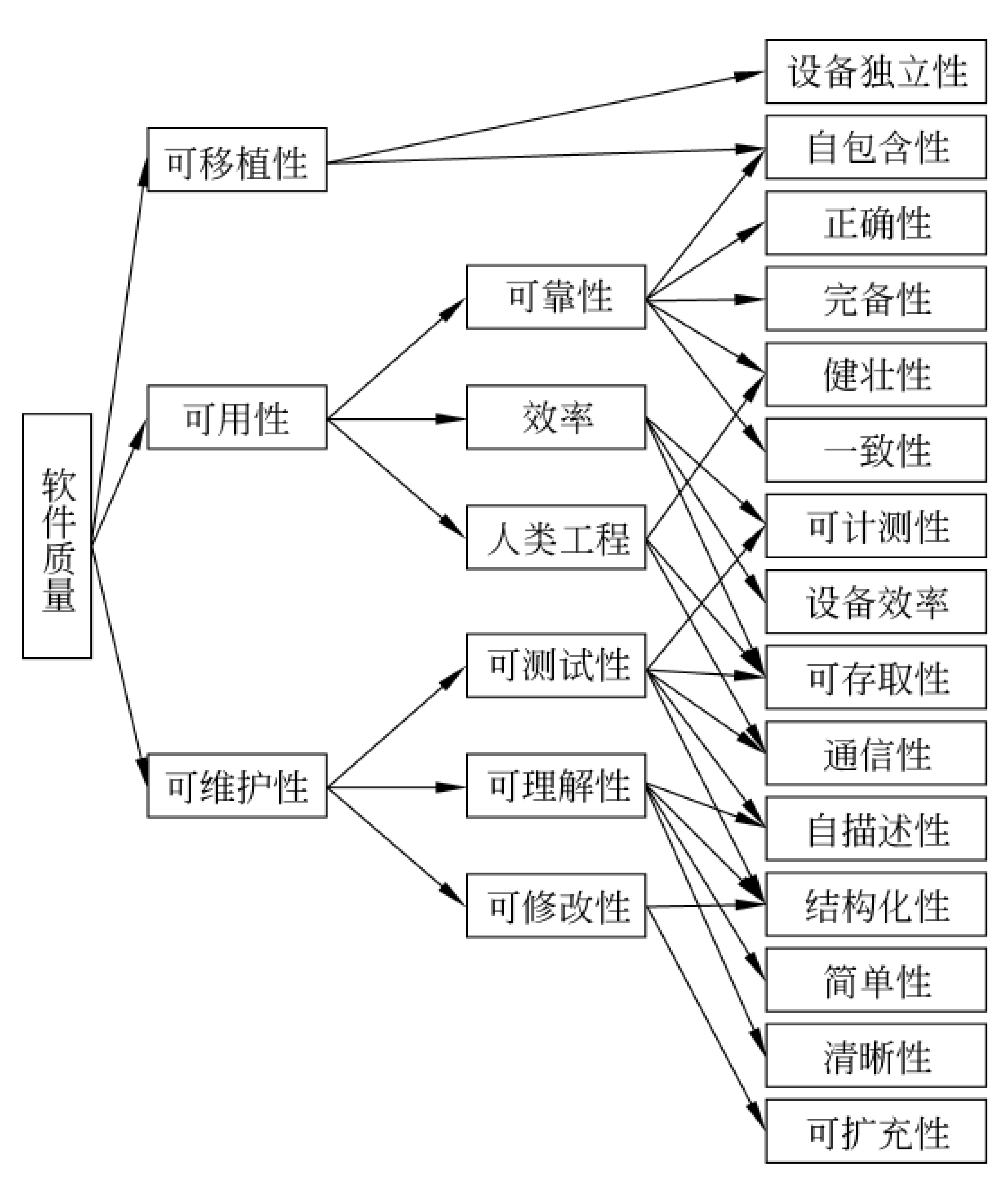


图 1-1 Bohm 质量模型

2. McCall 质量模型

McCall 质量模型是 1979 年由 McCall 等人提出的，它将软件质量的概念建立在 11 个质量特性之上，而这些质量特性分别是面向软件产品的运行、修正和转移的，如图 1-2 所示。

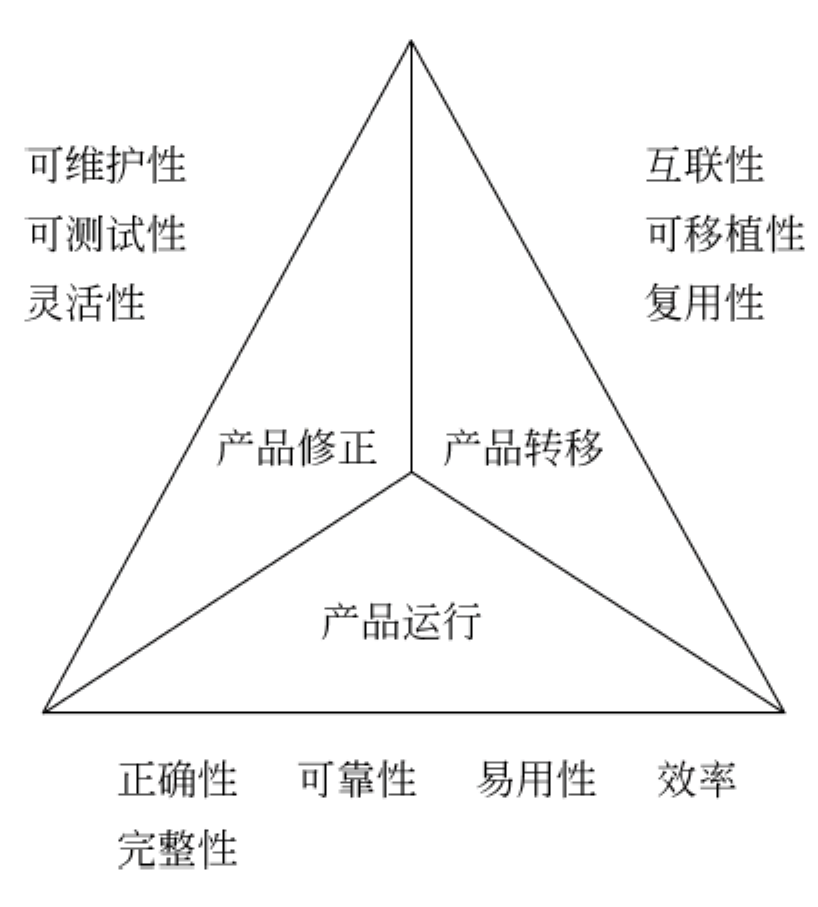


图 1-2　McCall 质量模型

3. ISO 的软件质量模型

按照 ISO/IEC 9126-1：2001，软件质量模型可以分为内部质量和外部质量模型、使用质量模型，而质量模型中又将内部和外部质量分成 6 个质量特性，将使用质量分成 4 个质量属性，具体见图 1-3 和图 1-4。

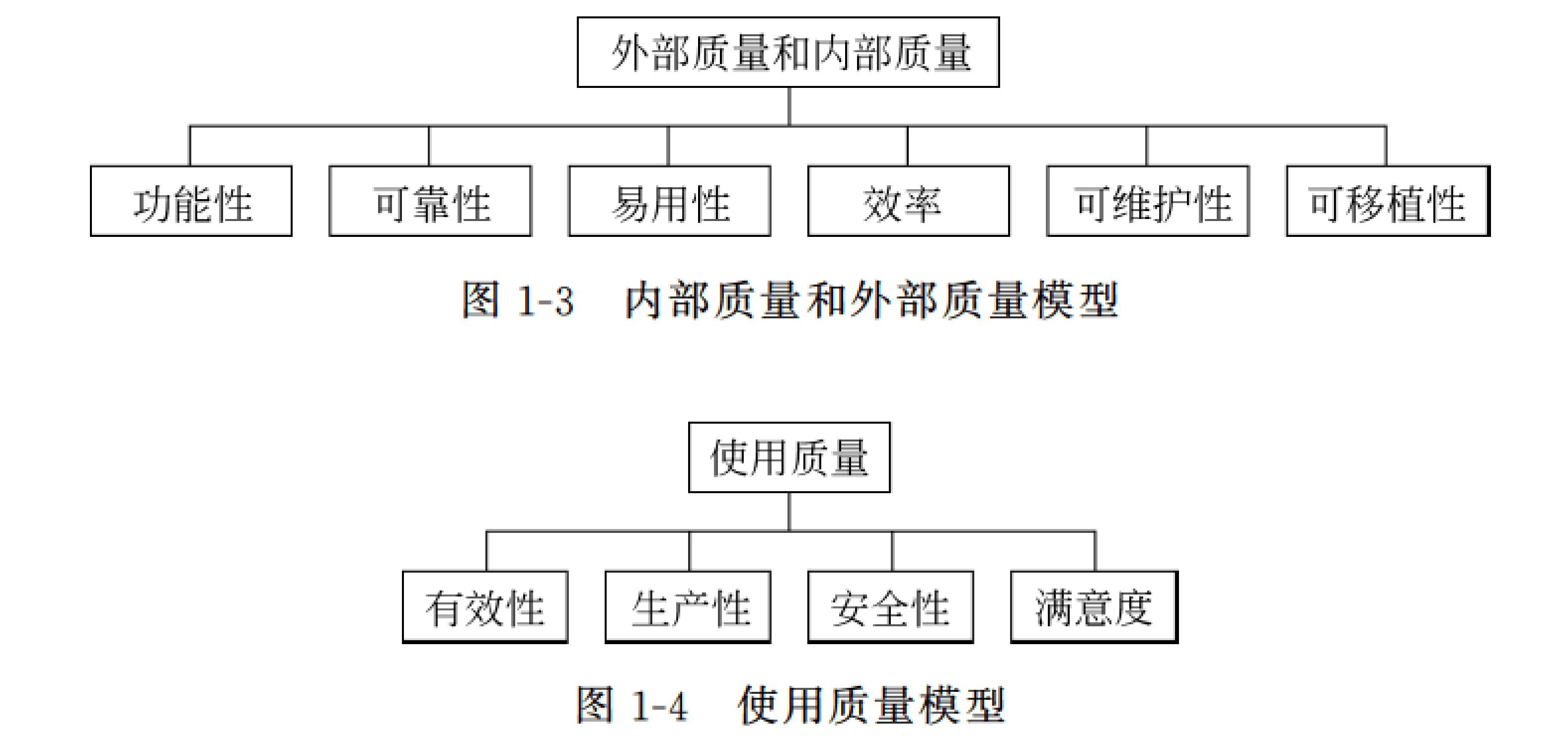


图 1-3　内部质量和外部质量模型

图 1-4　使用质量模型

1.1.4　软件质量的量度

软件质量的量度主要是根据软件生存周期中对软件质量的要求所进行的一项活动。它主要分为三方面：外部量度、内部量度和使用量度。

1. 外部量度

这是在测试和使用软件产品过程中进行的，通过观察该软件产品的系统行为，执行对其系统行为的测量得到量度的结果。

2. 内部量度

这是在软件设计和编码过程中进行的，通过对中间产品的静态分析来测量其内部质量特性。内部量度的主要目的是为了确保获得所需的外部质量和使用质量，与外部关系是二者相辅相成，密不可分。

3. 使用质量的量度

这是在用户使用过程中完成的，因为使用质量是从用户观点来对软件产品提出的质量要求，所以它的量度主要是针对用户使用的绩效，而不是软件自身。

1.2 软件测试的概念

1.2.1 软件测试的定义与目的

1. 软件测试的定义

Glenford J. Myers 于 1979 年给出测试的定义为软件测试是为发现错误而执行的一个程序或者系统的过程，同时给出了三个关于测试的重要观点：

测试是为了证明程序有错，而不是证明程序正确。

一个好的测试用例在于它能发现以前未发现的错误。

一个成功的测试是发现了以前未发现的错误的测试。

1990 年，IEEE 610.12 标准中给出测试的正式定义如下：

(1) 在规定条件下运行系统或构件的过程；

(2) 分析软件项目的过程。

2. 软件测试的目的

用最少的时间和人力，找出软件中潜在的各种错误和缺陷。软件测试的这一目的贯穿于整个测试过程中。

测试的另一收获是，它能够证明软件的功能、性能与需求说明相符合。

1.2.2 软件测试的原则

根据软件测试的目的，软件测试应该遵守以下原则：

(1) 应当把“尽早和不断地进行软件测试”作为软件开发人员的座右铭。

（2）测试用例应由测试的输入数据和与之对应的预期输出结果两部分组成。
（3）程序员应避免测试自己的程序。
（4）在设计测试用例时，应该包括合理的和不合理的输入条件。
（5）充分注意测试中的群集现象。
（6）严格执行测试计划，排除测试的随意性。
（7）应当对每一个测试结果做全面检查。
（8）妥善保存测试计划、测试用例、出错统计和最终分析报告。

1.3 软件的缺陷与错误

1.3.1 软件缺陷的定义和类型

所谓“缺陷（bug）”，即计算机软件或程序中存在的某种破坏正常运行能力的问题、错误，或者隐藏的功能缺陷。

软件缺陷的主要类型有：
（1）软件没有实现产品规格说明要求的功能。
（2）软件出现了不该出现的错误。
（3）软件实现了说明没提到的功能。
（4）软件没实现虽然规格说明中未明确提及但应实现的目标。
（5）软件难理解，不易使用。

1.3.2 软件缺陷的级别

软件缺陷有 4 种级别，分别为：

（1）致命的（Fatal）。致命的错误，导致系统或者应用程序崩溃、死机、系统悬挂，或者造成数据丢失、主要功能完全丧失。

（2）严重的（Critical）。功能或特性没有实现，主要功能部分丧失，次要功能完全丧失，或致命的错误声明。

（3）一般的（Major）。这种级别的错误不是很严重，虽然有一些缺陷，但是不影响系统和程序的基本使用。功能没有被很好地实现，没有达到预期要求。

（4）微小的（Minor）。无关紧要的小问题，软件仍然可以使用，不影响功能的实现。

除了严重性之外，还必须关注软件缺陷处于一种什么样的状态，以便跟踪和管理某个产品的缺陷，三种基本的缺陷状态包括：

（1）激活状态（Active 或 Open）。问题尚未解决，测试人员新报告的缺陷，或验证后缺陷仍然存在。

（2）已修正状态（Fixed 或 Resolved）。开发人员针对缺陷，修改程序，认为已解决问题，或者通过单元测试。

（3）关闭或非激活状态（Close 或 Inactive）。测试人员验证已修正的缺陷后，确认缺

陷不存在后的状态。

1.3.3 软件缺陷产生的原因

软件缺陷产生的原因主要有三个方面。

1. 技术问题

技术问题包括算法错误、语法错误、计算和精度错误、系统结构不合理、算法不科学、接口参数传递不匹配。

2. 团队工作

团队工作产生的原因包括：系统需求分析时对客户的需求理解不清楚，或者和用户沟通时存在困难；不同阶段的开发人员相互理解不一致；对于设计或者编程上的一些假定或依赖性，相关人员没有充分沟通。

3. 软件本身

软件本身的问题包括文档错误、内容不正确或者拼写错误；没有考虑大量用户的使用场合，从而可能会引起强度或负载问题；对程序逻辑路径或数据范围的边界考虑不够周全，漏掉某些边界条件，造成容量或边界错误；对一些实时应用，要精心设计和技术处理，保证精确的时间同步，否则容易引起时间上不协调、不一致带来的问题；没有考虑系统崩溃后的自我恢复或数据的异地备份、灾难性恢复等问题，从而存在系统安全性、可靠性的隐患；硬件或系统软件上存在的错误；软件开发标准或过程上的错误。

1.3.4 软件缺陷的分类

软件缺陷可分为5类。

1. 功能缺陷

功能缺陷包括规格说明书缺陷、功能缺陷、测试缺陷和测试标准引起的缺陷。

2. 系统缺陷

系统缺陷包括外部接口缺陷、内部接口缺陷、硬件结构缺陷、操作系统缺陷、软件结构缺陷、控制与顺序缺陷、资源管理缺陷。

3. 加工缺陷

加工缺陷包括算术与操作缺陷、初始化缺陷、控制和次序缺陷、静态逻辑缺陷。

4. 数据缺陷

数据缺陷包括动态数据缺陷、静态数据缺陷、数据内容、结构和属性缺陷。

5. 代码缺陷

包括数据说明错、数据使用错、计算错、比较错、控制流错、界面错、输入和输出错等。

1.3.5 修复软件缺陷的代价

缺陷发现或解决得越迟，成本就越高。平均而言，如果在需求阶段修正一个错误的代价是1，那么，在设计阶段就是它的3～6倍，在编程阶段是它的10倍，在内部测试阶段是它的20～40倍，在外部测试阶段是它的30～70倍，而到了产品发布出去时，这个数字就是40～1000倍，修正错误的代价不是随时间线性增长的，而几乎是呈指数增长的。

1.4 软件测试的经济学与心理学

1.4.1 软件测试的心理学

1. 程序测试的过程具有破坏性

软件测试适合于被视作发现程序中错误的破坏性过程。一个成功的测试，通过诱发程序发生错误，可以在这个方向上促进软件质量的改进和提高。

2. 程序员应避免测试自己的程序

开发和测试是不同的活动，开发是创造或建立某种事物的行为，而测试是证实其不正常，一个人不太容易同时做好这两项工作。由于程序员主观上对问题的叙述和说明的误解而产生的错误在程序员测试自己的程序时，往往还会带来同样的误解从而导致问题难以被发现。

3. 程序设计组织不应测试自己的程序

要程序设计组织在测试自己的程序时保持客观的态度是困难的，因为如果用正确的定义看待测试，就不大可能按预定计划完成测试，也不大可能把消耗的代价限制在消耗的范围以内。

1.4.2 软件测试的经济学

为了应对测试经济学的挑战，应该在开始测试之前建立某些策略。黑盒测试和白盒测试是两种普遍的策略。

1. 黑盒测试

黑盒测试是一种重要的测试策略，又称为数据驱动的测试或输入输出驱动的测试。使用这种测试方法时，将程序视为一个黑盒子，测试目标与程序的内部机制和结构完全无关，而是将重点集中放在发现程序不按其规格说明书正确运行的环境条件上。在这种方法中，测试数据完全来源于软件规格说明，不需要了解程序的内部结构。

2. 白盒测试

白盒测试又称为逻辑驱动测试，这种测试策略是对程序的逻辑结构进行检查，从中获取测试数据。

1.5 软件质量保证

1.5.1 软件质量保证概要

软件质量保证是通过对软件产品有计划地进行各种评审和审核，从而验证和确认生成出来的软件产品是否符合标准的系统工程。

美国CMU SEI制定的SW－CMM（软件能力成熟度模型）在SQA KPA（软件质量保证关键过程域）中规定软件质量保证（Software Quality Assurance，SQA）活动的目标有以下几个：

（1）制定和规划软件质量保证的任务。

（2）客观地验证软件产品和各项任务是否遵循适用的标准、规程和需求。

（3）相关小组和个人保持良好的沟通，及时通知他们在软件质量保证方面的结果。

（4）高层管理人员能够参与并帮助解决项目中不能解决的不相容问题。

1.5.2 软件质量保证活动的实施

（1）目标（Target），这一步主要是设定质量特性与质量子特性的评价标准。

（2）计划（Plan），这一步确定适合于被开发软件的各个阶段、各个活动的质量评测检查项目与质量量度方法。同时，还要研讨实现质量目标的方法与手段。

（3）实施（Do），这一步是在软件开发的过程中，参与各个活动的评审和各个阶段的正式技术评审。对软件开发中各个阶段的进展、完成质量及出现的问题进行监督，审核各项活动和中间产品的生成是否遵循相应的过程规范，形成报告。

（4）检查（Check），以计划阶段确定的质量量度标准进行评价，看质量是否合格。

（5）行动（Action），对评价发现的问题进行改进活动。

1.5.3 SQA 与软件测试的关系

SQA 与软件测试的区别在于：软件质量保证是采取一些措施或方法来改进软件开发过程，尽量防止软件缺陷的产生；而软件测试是尽可能地发现软件缺陷并确保缺陷得以改正，使得软件产品更加健壮。

SQA 与软件测试的联系是：二者相互依赖，相互促进。软件测试人员也会做一些质量保证的工作，这主要表现在上面活动的实施阶段，软件质量保证人员也会从事一些测试活动，主要侧重于对测试过程执行的验证和确认。

1.6 本章小结

软件质量是产品、组织和体系或过程的一组固有特性，反映它们满足顾客和其他相关方面要求的程度。软件质量有三种模型：Bohm 质量模型、McCall 质量模型、ISO 的软件质量模型。软件质量的量度主要分为三方面：外部量度、内部量度和使用量度。本章还阐述了软件测试的定义、目的和要遵守的原则，软件缺陷的产生原因和分类，心理学与经济学对软件测试的影响以及软件质量保证的目标及实施和 SQA 与软件测试的关系。

第 2 章　软件测试类型及其在软件开发过程中的地位

2.1　软件开发阶段

2.1.1　软件生存周期

软件生存周期的主要阶段是制订计划、系统与软件需求定义、软件设计、编程与单元测试、集成测试与系统测试、运行和维护。各阶段的主要任务如下所述。

1. 制订计划

确定要开发软件的总目标，研究该任务的可行性，探讨解决问题的方案，估计成本效益和进度，制订实施计划及可行性研究报告，提交管理部门审查。

2. 系统与软件需求定义

基于各种方式获取的需求和建立的业务对象模型和分析模型，编写系统和软件需求规格说明，提交管理机构进行需求评审。

3. 软件设计

这是软件工程的核心，主要分为概要设计和详细设计。

4. 编程和单元测试

将软件设计规格说明转换为计算机可接受的程序代码，即编程实现和单元测试的任务。

5. 集成和系统测试

对已测试过的模块进行组装，进一步进行测试。

6. 运行和维护

将软件投入使用，若发现问题，应适当进行更正。

2.1.2　软件测试的生存周期模型

软件测试直观上讲仅是对测试对象进行检查、验证，似乎很简单，但实际上软件测试是有其严格测试过程的，图 2-1 给出了软件测试的生存周期模型，描述了软件测试的全过程。

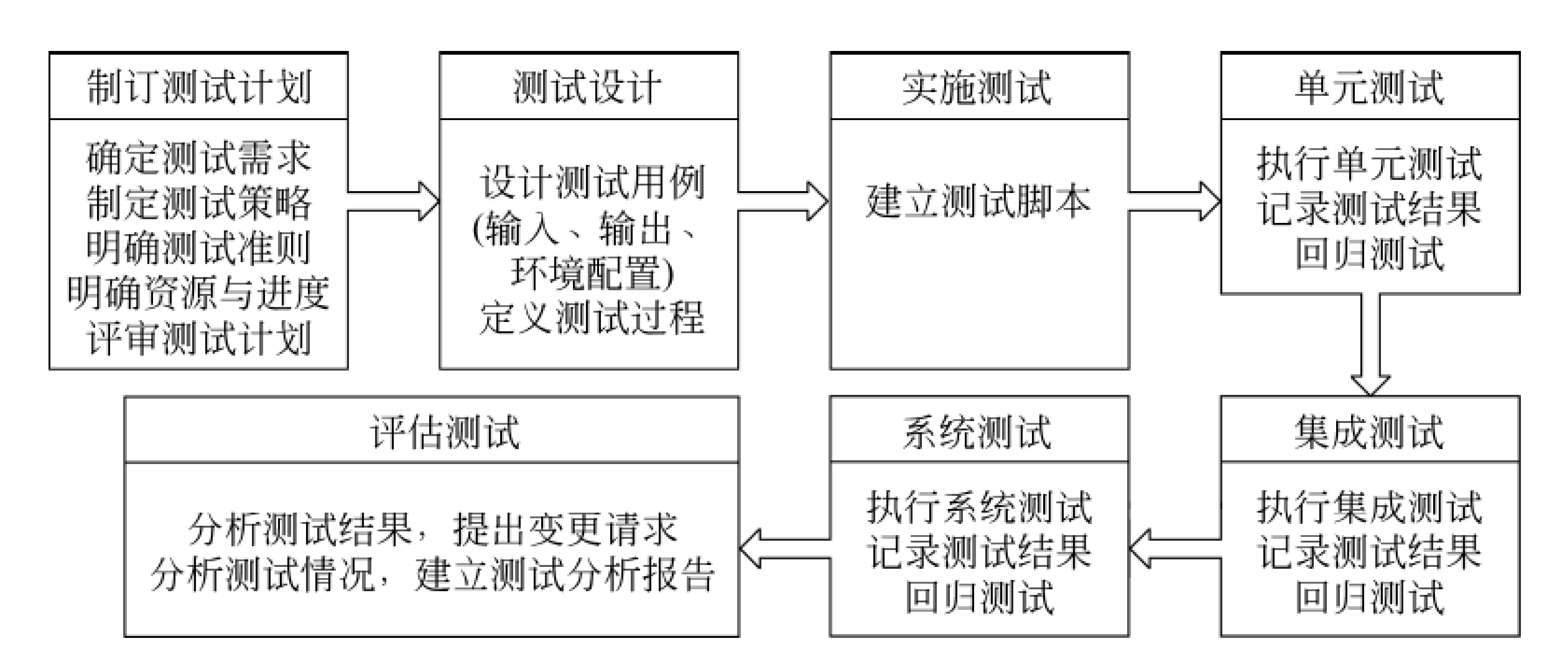


图 2-1　软件测试的生存周期模型

2.1.3　测试信息流

测试过程主要有三种输入：

(1) 软件配置。包括软件需求规格说明、软件设计规格说明、源代码等。

(2) 测试配置。包括表明测试工作如何进行测试计划、给出测试数据的测试用例、控制测试进行的测试程序等。

(3) 测试工具。为了提高如今的测试效率，需要有测试工具的支持，它为测试提供某种服务，减轻人们完成测试任务中的手工劳动。

测试工作结束之后，还要对所有的测试结果进行比较，然后开始排错(调试)。

2.2　规划阶段的测试

2.2.1　目标阐述

目标阐述又称为问题定义。规划人员在此阶段首先描述自己对产品的构想——产品应该做什么及为什么要做。此时形成的文档很简单，只大概地描述软件的界面及性能等。最终实现的产品也不一定会满足所有的目标。目标阐述的意义在于为开发团队提供一个共同努力的方向。

2.2.2　需求分析

需求是指必须实现的目标。规划人员在此阶段应将需求转化成功能性的术语，应将如何设计和实现的细节推迟到后续阶段留给开发人员完成。他们必须详细地规定功能、性能、可靠性目标及用户界面的某些方面。

2.2.3 功能定义

功能定义在需求分析文档和设计文档之间架起了沟通的桥梁。功能定义将软件需求转化为一系列特性和功能。它包含的信息仅够程序员明白所描述的内容，一般不会规定实现这些特性的细节。

2.2.4 规划阶段进行的测试

在规划阶段里，测试的对象是规划人员的构想，而不是代码。测试人员（即评审人员）包括了营销人员、产品经理、设计人员和人类工程分析师。

2.3 设计阶段的测试

2.3.1 外部设计

主要是从用户的角度对产品进行描述，主要是设计用户界面。它描述了所有的屏幕显示和其他输出、可用的命令、命令语法、命令或特性之间的联系，以及对所有可能输入的反应。

外部设计规格说明是在外部设计期间产生的文档，这是测试阶段很重要的依据。

用户手册是另一种文档，与外部设计规格说明不同的是，它是在需求获取与定义阶段就开始建立，以后要不断细化和完善的文档。

外部设计在后期会被多次修改，这是因为很难设计出一个完美无缺的用户界面。如果用户不能够方便地使用程序，那么外部设计就应当被视为彻底失败。

2.3.2 内部设计

主要描述产品的内部工作机制，它细化软件的功能、性能以及其他软件特性，并把它们分解到不同的程序单元中，同时设计系统功能所需的数据内容、数据结构和数据流，以及程序代码如何运行（逻辑设计）。内部设计具体又可以细分为结构设计、数据设计和逻辑设计。

外部设计和内部设计并行开展、相互制约、相互要求。

2.3.3 设计阶段的测试

在设计阶段，测试的对象来自设计文档，主要采用的是评审方式。这里的文档主要包括：

- 外部设计（用户界面设计，与其他元素的接口设计，系统构件部署设计）的规格

说明。

- 内部设计(功能设计,系统体系结构设计,数据设计)的规格说明。
- 逻辑设计(模块算法与数据结构设计)的规格说明。

2.3.4 伪代码分析

设计人员在进行逻辑设计时使用伪代码描述非常方便。如果使用的是伪代码的正式定义版本,就可以采用伪代码分析器来查找设计中存在的缺陷。伪代码分析器的工作依赖于输入所有即将编写的代码完备的底层(逻辑)设计。

2.4 编程阶段的测试

2.4.1 白盒测试与黑盒测试

白盒测试是编程阶段中最有效的测试类型,程序员主要采用白盒测试对每个程序单元进行测试,因为它是这个期间程序员使用最有效的测试类型。

白盒测试与黑盒测试的区别是:白盒测试是在程序员十分了解程序的前提下,对程序的逻辑结构进行的测试。而黑盒测试则将程序视为一个黑盒子,测试人员提供输入数据,观察输出数据,并不了解程序是如何运行的。

在白盒测试中,程序员要深入程序中以开发测试,这样带来的好处体现在重点测试、测试覆盖、控制流、数据完整性、内部边界、特定算法测试等方面。

2.4.2 结构测试与功能测试

结构测试属于白盒测试,关注的是如何选择合适的程序或子程序路径来执行有效的检查。

功能测试则属于黑盒测试,对功能的测试通常通过提供输入数据,检查实际的输出结果,很少考虑程序的内部结构。

2.4.3 路径测试:覆盖准则

路径测试属于白盒测试,是在程序控制流程图的基础上,通过分析控制构造的环境复杂性,导出基本的执行路径集合,从而设计测试用例的方法。常用的覆盖准则有语句覆盖、分支覆盖和条件覆盖等,这些准则对后面测试用例方法的设计有重要作用。

2.4.4 增量测试与大突击测试

增量测试(Incremental Testing)是指将程序模块逐步集成进行测试。而大突击测试

(Big Bang Testing)是指将程序成块地进行测试,这两种测试方法在后面的集成测试和系统测试中常用。

2.4.5 自顶向下测试与自底向上测试

这两种测试方法皆属于增量测试。自底向上测试是首先测试最底层的模块,利用辅助的驱动模块调用,然后测试高层次的模块。而自顶向下测试则刚好相反,它首先测试顶层模块,无须编写驱动模块,但要使用桩模块,然后测试下一层模块。

2.4.6 静态测试与动态测试

静态测试(或称静态分析)不必运行程序,目的是收集有关程序代码的结构信息而非查错。而动态测试则需要运行程序,目的是查错而非检查程序代码的结构信息。

2.4.7 性能测试

性能测试的主要目标是发现和改正性能缺陷并提高性能,大多数通过黑盒测试来实现性能测试。

2.5 回归测试

回归测试是指:一经发现并改正了程序中隐藏的缺陷,然后再重新执行以前发现这个缺陷的测试,查看此缺陷是否重现。另外,当对发现的缺陷进行修改之后,执行一系列基准测试,以确认程序的修改没有对其他部分产生干扰,这也称为回归测试。

2.6 运行和维护阶段的测试

在运行和维护阶段,需要在运行环境中对软件产品进行性能监视,可能还要进行相应的修改。

2.7 本章小结

本章讨论了软件的开发阶段,介绍了规划阶段的测试、设计阶段的测试、编程阶段的测试、回归测试以及运行和维护阶段的测试。

第 3 章　代码检查、走查与评审

3.1　桌上检查

3.1.1　桌上检查的检查项目

桌上检查是一种程序员检查自己的源程序的方法。桌上检查的目的是发现程序中的错误。桌上检查的检查项目主要有检查变量、标号的交叉引用表，检查子函数、宏、函数，等价性检查，常量检查，标准检查，风格检查，比较控制流，选择、激活路径，补充文档等。

3.1.2　对程序代码做静态错误分析

桌上检查的静态分析分为两部分：生成引用表和进行静态错误分析。

1. 生成各种引用表

引用表是为了支持后面对源代码进行静态检查，按功能不同可分为三类：

可直接从表中查出说明/使用错误，如循环层次表、变量交叉引用表、标号交叉引用表等；

为用户提供辅助信息，如子函数（宏、函数）引用表、等价（变量）表、常数表等；

用来做错误预测和程序复杂度计算，如操作符和操作数的统计表。

2. 进行静态错误分析

主要是用于确定在源程序中是否存在错误或危险结构，通常有以下几种：

- 类型和单位分析；
- 引用分析；
- 表达式分析；
- 接口分析。

3.2　代码检查

3.2.1　特定的角色和职责

代码检查小组通常规模很小，一般人数为 4～7 人不等。规模大的代码检查小组主要检查文档，小规模的代码检查小组主要检查具体技术的实现。小组人员的角色分配通常如下：

1. 协调人员

支持引导代码检查的执行过程，全面负责代码检查工作。协调人员通常是由开发部门挑选和培训的，并负担指定开发项目的代码检查工作。

2. 开发人员

是检查项目的生产者，主要负责提供检查项目资料和回答检查人员的问题，通常开发人员也是代码检查的检查人员。

3. 检查人员

检查小组每一个人都可以认为是检查人员，可兼任不同的角色。

4. 讲解员

负责在检查会议中讲解检查项目，引导小组对产品进行彻底检查。最佳人选是相关文档和程序代码的编写者。

5. 记录员

负责会议期间在检查表上记录发现的每一个错误，同时也承担作为一般检查人员的任务。

3.2.2 代码检查过程

代码检查(Code Inspection)，就是以小组为单位阅读代码，应用一系列规程和错误检查技术，检查实际的产品，发现错误和缺陷的过程。

3.2.3 用于代码检查的错误列表

代码检查的重要部分就是对照错误列表来检查常见的错误，错误列表是独立于一般的编程语言的，主要包括：

- 数据引用错误；
- 数据声明错误；
- 运算错误；
- 比较错误；
- 控制流程错误；
- 接口错误；
- 输入输出错误；
- 其他检查。

3.3 走查

3.3.1 特定的角色和职责

走查小组由3～5人组成,至少包括的成员如下:
(1) 扮演类似代码检查过程中"协调人"的角色一人;
(2) 负责记录所有查出的错误的记录员一人;
(3) 测试员一人。
建议另外的参与者有:
(1) 一位有经验的程序员;
(2) 一位程序员新手;
(3) 一位最终将维护程序的人员;
(4) 一位来自其他不同项目的人员;
(5) 一位来自该软件编程小组的程序员。

在走查前,走查小组的各成员需要评审相关资料,在走查过程中,由于参与审查的人员中只有一人是程序编写者,因此走查的主要工作是由其他人而非程序编写者完成的。走查小组秘书的职责是负责记录走查期间做出的所有说明,包括发现的问题、样式方面的错误、遗漏、矛盾、改进建议或者替换解决方法。

3.3.2 走查的过程

1. 计划走查会议

协调人员应完成下面的工作:
(1) 选择一名或多名人员组成走查小组。
(2) 安排走查会议的时间和地点。
(3) 分发所有必需的材料给审查人员。

2. 走查产品

走查人员负责为走查做准备,并且在需要的时候要完全熟悉标准、检查表和其他任何提供的用于走查的信息。走查人员走查产品并且准备在走查会议上讨论他们对产品做出的评注、建议、问题。

3. 执行走查

走查小组开会,集体扮演计算机角色,让事先准备好的测试用例沿程序的逻辑运行一遍,随时记录程序的踪迹,供分析和讨论用。每个测试用例都在人们脑中进行推演。

走查的优点有:

(1) 能在代码中对错误进行精确定位,降低调试成本;
(2) 可以发现成批的错误,便于一同得到修正。

4. 解决缺陷

程序员和走查人员解决走查中发现的问题。

5. 走查记录

记录走查人员的名字、被审查的产品、走查的日期、缺陷、遗漏、矛盾和改进建议列表。

6. 产品返工

根据走查的记录,程序员更新产品,纠正所有的缺陷、遗漏、效率问题和改进产品。

3.3.3 走查中的静态分析技术

走查着重从流程的角度考察程序,借助程序流程图或调用图对数据流和控制流进行静态分析,调用图中,节点表示程序单元,有向边表示程序单元之间的控制和调用,通过调用图可以检查程序中变量的说明和引用,全局变量、参数误用的问题,同时还为动态测试用例的设计提供可靠的依据,在调用图中是不能对程序进行修改的。

3.4 同行评审

3.4.1 为什么需要评审

同行评审(Peer Review)是一种通过作者的同行来确认缺陷和需要变更区域的检查方法。评审的目的是发现产品的缺陷,因此在评审上的投入可以减少大量的后期返工。同行评审的作用非常突出,既缩减了工作时间,又节约了大量成本。及时进行同行评审不仅有利于提高软件的质量,而且可以进一步提高工程师的工作效率。

3.4.2 同行评审的角色和职能

同行评审的过程主要由评审小组组织和进行。一个评审小组主要由以下角色构成。

(1) 协调人(评审组长):和作者共同商讨决定具体的评审人员;安排正式的评审会议;与所有评审人员举行一个准备会议,确保所有的评审员都明确他们的角色和责任;确保会议的输入文件都符合要求;如果作者或者评审员没有为即将召开的评审会议做好充分的准备,则需要重新安排会议并通知大家;确保大家的关注点都是评审内容的缺陷;确保所有提出的缺陷都被记录下来;跟踪问题的解决情况;和项目组长沟通评审的结果。

(2) 作者:确保即将评审的文件已经准备好;与项目组长、协调人一起定义评审小组的成员。

(3) 评审员(读者):必须具备良好的个人能力。通常在评审员的选择上应该包含上一级文档的作者代表和下一级文档的制定作者。

3.4.3 同行评审的内容

同行评审涉及的内容很多,主要可以分为4类。

1. 管理评审

管理评审是对项目管理体系的适应性和管理活动的有效性进行评价。评审结果是提交管理评审报告。

2. 技术评审

技术评审是对产品以及各阶段的输出内容进行评估。目的是确保需求说明、设计说明书与用户需求一致,并按计划对软件进行正确的开发。技术评审的对象主要包括:需求文档,源代码,测试用例等,评审检查列表、其他必需的文档等。

3. 文档评审

在软件开发过程中,需要进行评审的文档很多,主要包括:需求评审(用户需求规格说明、产品需求规格说明、功能规格说明等),设计评审(软件总体设计规格说明、详细设计规格说明等),代码评审,质量验证评审(测试计划、测试用例等)。

4. 过程评审

这里的过程是指软件开发过程。过程评审的主要任务是通过对流程的控制,保证SQA组织定义的软件过程在项目中得到遵循,同时保证质量、保证方针能更快更好地执行。

3.4.4 评审的方法和技术

同行评审方法很多,基于正式化程度可以分为以下几种。

1. 临时评审

指一程序员临时请另一程序员用几分钟检查一个缺陷,这是最不正式的检查方法。

2. 桌上检查或轮查

这种检查方法有时称为分配审查方法,是一种由多人组成的并行的同行桌上检查。

3. 结对评审

又称同行桌上检查或伙伴检查。桌上检查是指作者自己检查源代码,而结对评审就

是除作者外，再请一位评审员对产品进行的桌上检查。这是最便宜的评审方法。

4. 走查

走查是一种非正式的评审，方法主要有两种：一是使用样品数据做测试用例，二是按脚本执行，通过脚本描述具体场景，说明系统如何在交互中完成预定功能。

5. 小组评审

是一种“轻型”的审查，有计划和结构，非常接近正式评审，适用于不需要严格审查过程的工作产品。

6. 正式审查

与上面的小组评审很相似，但比它更严格，是最系统、最严密也最实用的评审方法。

3.5 本章小结

本章阐述了桌上检查对代码静态错误分析、代码检查小组成员及其职责划分、代码检查的错误列表、走查的步骤、走查中的静态分析技术、同行评审的角色和职责、同行评审的内容以及同行评审的方法和技术。

第4章　覆盖率测试

4.1　覆盖率概念

覆盖率是量度测试完整性的一个工具，通常可以分为逻辑覆盖和功能覆盖。覆盖率的计算公式如下：

覆盖率 = 被执行到的项数 / 总项数 × 100%

公式中的"项"视不同情况而定，对于具体准则可定义它的语义。

覆盖率对软件测试有着非常重要的作用。覆盖率数据可以指导人们设计能够增加覆盖率的测试用例。这样就能有效地提高测试质量，避免设计无效的测试用例。

4.2　逻辑覆盖

逻辑覆盖是以程序内部的逻辑结构为基础设计测试用例的技术，属于白盒测试。根据覆盖率的不同，又可以分为语句覆盖、判定覆盖、判定/条件覆盖、条件组合覆盖和路径覆盖。

4.2.1　语句覆盖

语句覆盖就是指设计若干测试用例，运行被测程序，使每个可执行语句至少执行一次。它是最弱的逻辑覆盖准则。指令块覆盖是语句覆盖的一个变体，唯一的区别是计算方式不同。指令块是函数内部的一组语句，在这组语句中不存在控制语句，计算公式如下：

语句覆盖率 = 被评价到的语句数量 / 可执行的语句数量 × 100%

指令块覆盖率 = 被执行的指令块数量 / 程序中的指令块总数 × 100%

4.2.2　判定覆盖

判定覆盖，有时也称为分支覆盖，就是指设计若干测试用例，运行被测程序，使得每个判定的取真分支和取假分支至少评价一次。判定路径覆盖是判定覆盖的一个变体，这里的判定路径是指一个语句序列，其起始位置是程序入口或一个判定的开始，结束位置是下一个判定的开始或程序出口，计算公式如下：

判定覆盖率 = 被评价到的判定分支个数 / 判定分支的总数 × 100%

判定路径覆盖率(DDP) = 被评价到的判定路径数量 / 判定路径的总数 × 100%

4.2.3 条件覆盖

条件覆盖就是指设计若干测试用例，运行被测程序，使得每个判定的每个条件的可能取值至少评价一次，计算公式如下：

条件覆盖率 = 被评价到的条件取值的数量 / 条件取值的总数 × 100%

4.2.4 条件/判定覆盖

条件/判定覆盖就是指设计足够的测试用例，使得判定语句的每个条件的所有可能取值至少评价一次，同时每个判定语句本身的所有可能分支也至少评价一次，计算公式如下：

条件/判定覆盖率 =被评价到的条件取值和判定分支的数量 /（条件取值总数 + 判定分支总数）× 100%

4.2.5 条件组合覆盖

条件组合覆盖就是指设计足够的测试用例，使得每个判定的所有可能条件取值至少评价一次，计算公式如下：

条件组合覆盖率 = 被评价到的条件取值组合的数量/条件取值组合的总数 × 100%

4.2.6 路径覆盖

路径覆盖就是指设计足够的测试用例，执行程序中所有可能的路径。基于路径覆盖的测试是最强的覆盖测试，但是路径覆盖并不一定能包含判定/条件覆盖，计算公式如下：

路径覆盖率 = 被执行到的路径数量/程序中的路径总数 × 100%

4.2.7 ESTCA 覆盖

Foster 通过大量的实验确定了程序中谓词最容易出错的部分，得出一套错误敏感测试用例分析规则：

规则 1　对于 A rel B(rel 可以是<、=和>)型的分支谓词，应适当地选择 A 与 B 的值，使得测试执行到该分支语句时，$A<B$、$A=B$、$A>B$ 的情况分别出现一次。

规则 2　对于 A rel C(rel 可以是<或>，A 是变量，C 是常量)型的分支谓词，当 rel 为<时，应适当地选择 A 的值，使得 $A=C-M$(M 是距 C 最小的容许正数，若 A 和 C 均为整型时，$M=1$)。同样，当 rel 为>时，应适当地选择 A 的值，使得 $A=C+M$。

规则 3　对外部输入变量赋值，使其在每一测试用例中均有不同的值与符号，并与同一组测试用例中其他变量的值与符号不一致。

4.2.8 LCSAJ 覆盖

一个 LCSAJ 是一组顺序执行的代码，以控制流跳转为其结束点，它的定义如下：

它起始于程序的入口或者一个可能导致控制流跳转的点；

它结束于程序的出口或者一个可能导致控制流跳转的点；

对于该点，一个跳转在后面的序列中产生。

LCSAJ 覆盖准则是一个分层的覆盖准则：

【第一层】语句覆盖；

【第二层】分支覆盖；

【第三层】LCSAJ 覆盖；

【第四层】两两 LCSAJ 覆盖；

⋮

⋮

【第 $n+2$ 层】每 n 个首尾相连的 LCSAJ 组合在测试中都要经历一次。

4.3 路径测试

4.3.1 分支结构的路径测试

分支结构有两种：嵌套型分支结构和串联型分支结构。

对于嵌套型分支结构，若有 n 个判定语句，则存在 $n+1$ 条不同的路径，需要 $n+1$ 个测试用例来覆盖它的每一条路径。

对于串联型分支结构，若有 n 个判定语句，则存在 $2n$ 条不同的路径，因此需要 $2n$ 个测试用例来覆盖它的每一条路径。当 n 较大时，路径数会达天文数字，无法完成测试。此时为减少测试用例数目，可以采用正交实验设计法来设计测试用例，测试路径数目可从 $t=2n$ 减到 $n+1\sim 2n$ 条。正交实验设计法的具体步骤如下：

(1) 设串联型分支结构中有 n 个判定语句，计算满足关系式 $n+1\leqslant 2^m$ 的最小自然数 m。

(2) 设 $t=2^m$，取正交表 L_t，并利用它设计测试数据。

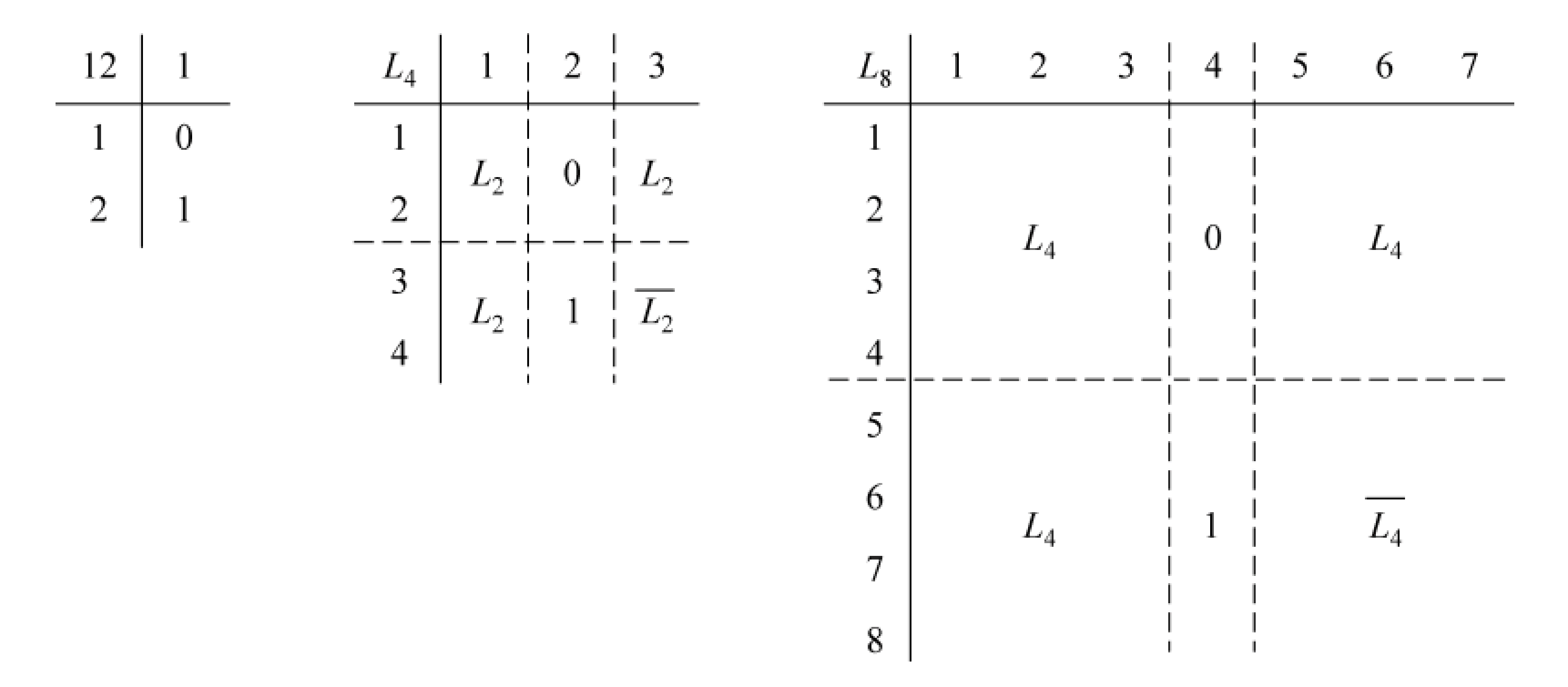
其中正交表的构造方法如图 4-1 所示。

4.3.2 循环结构的路径测试

循环分为 4 种不同类型：简单循环、嵌套循环、连锁循环和非结构循环，见图 4-2。

1. 简单循环

对于简单循环，测试应包括以下几种(n 表示循环允许的最大次数)。



12	1
1	0
2	1

L_4	1	2	3
1, 2	L_2	0	L_2
3, 4	L_2	1	$\overline{L_2}$

L_8	1 2 3	4	5 6 7
1, 2, 3, 4	L_4	0	L_4
5, 6, 7, 8	L_4	1	$\overline{L_4}$

图 4-1 正交表的构造方法

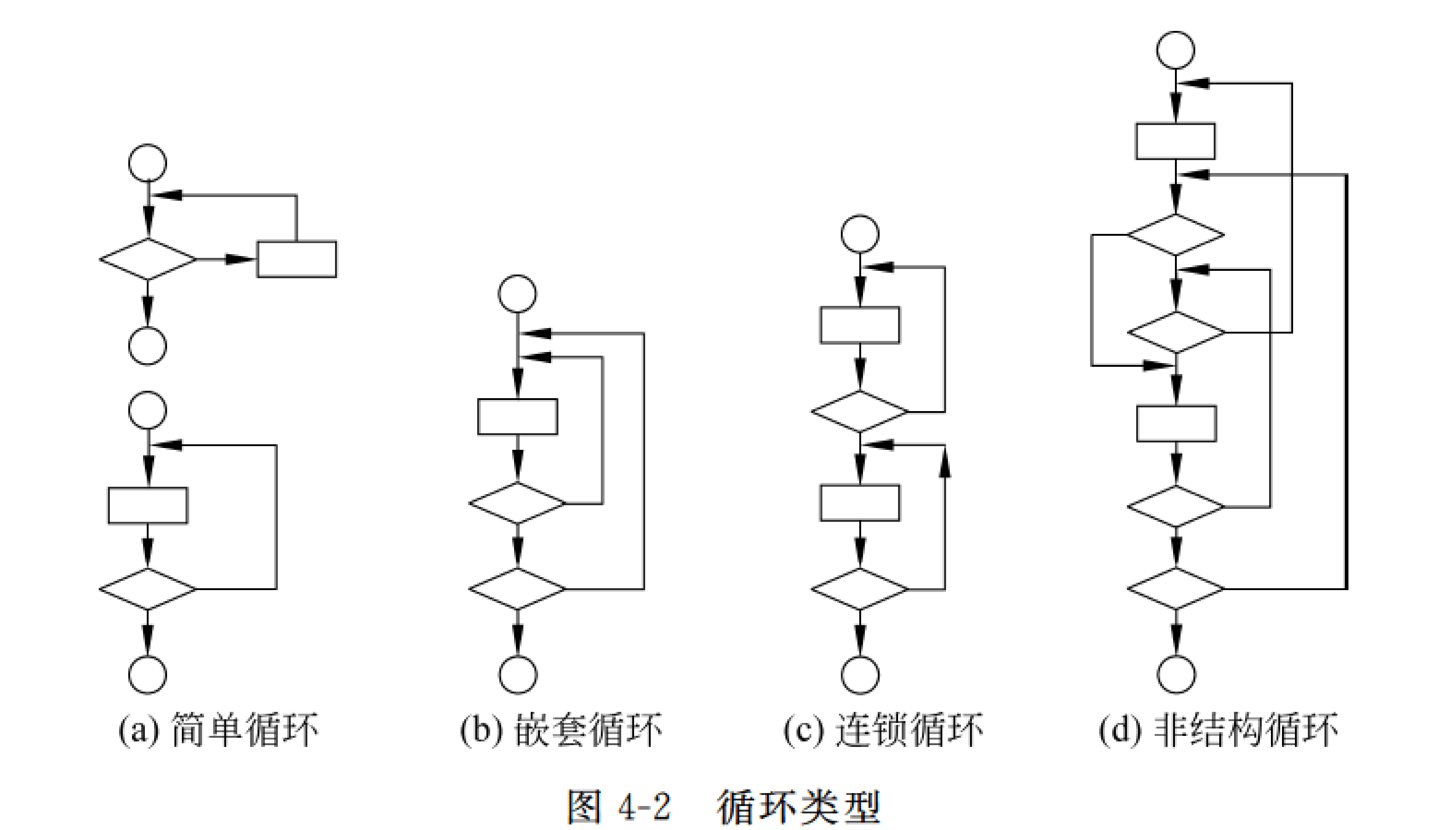


图 4-2 循环类型

(1) 0 次循环：从循环入口直接到出口；

(2) 1 次循环：查找循环初始值方面的错误；

(3) 2 次循环：检查多次循环时才能暴露的错误；

(4) m 次循环：$m<n$，也是检测在多次循环时才能暴露的错误；

(5) 最大次数循环、比最大次数多一次的循环、比最大次数少一次的循环。

2. 嵌套循环

对于嵌套循环，给出一种有助于减少测试数目的测试方法如下：

(1) 除最内层循环外，从最内层循环开始，置所有其他层的循环为最小值。

(2) 最内层循环做简单循环的全部测试。

(3) 逐步外推，对其外面一层循环进行测试。

(4) 反复进行，直到所有各层循环测试完毕。

(5) 对全部各层循环同时取最小循环次数，或者同时取最大循环次数。

3. 连锁循环

若各个循环相互独立，则连锁循环测试方法同简单循环；若几个循环不是互相独立的，则测试方法同嵌套循环。

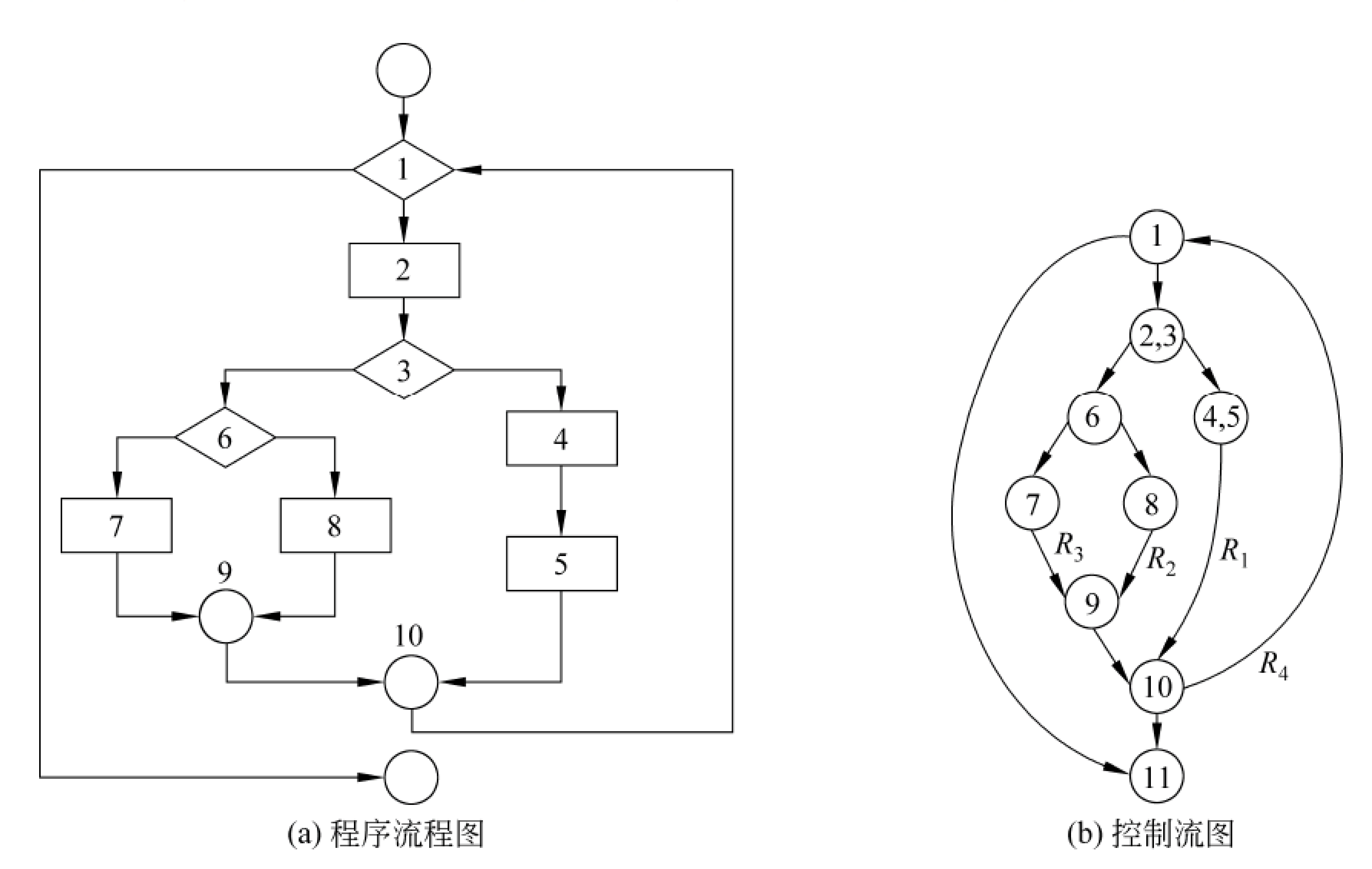
4. 非结构循环

对于非结构循环，要使用结构化程序设计方法重新设计测试用例。

4.3.3 Z 路径覆盖与基本路径测试

1. 程序的控制流图

控制流图是描述程序控制流的一种图示方法，其中控制流图中的箭头称为边，表示控制流的方向，一条边必须终止于一个节点，边与节点圈定的空间称为区域，当对区域计数时，图形外的空间也应记为一个区域。程序流程图可以转换为控制流图，如图 4-3 所示。



(a) 程序流程图　　(b) 控制流图

图 4-3　程序流程图和控制流图

由控制流图可以得到区域数为 4。

2. 程序的环路复杂性

环路复杂性（又称圈复杂度）是对程序逻辑结构所做的一种定量量度。环路复杂性的求法如下：

① 将环路复杂性定义为控制流图中的区域数。

② 控制流图 G 的环路复杂性记为 $V(G)$，则 $V(G)=E-N+2$，其中 E 为边数，N 为图中节点总数。

③ $V(G)=P+1$,其中 P 表示控制流图中的判定节点数。

3. 基本路径测试方法设计测试用例

基本路径测试是在控制流图的基础上,通过分析环路复杂性,导出基本可执行路径的集合,从而设计测试用例的方法。描述这种路径覆盖的准则就是 Z 路径覆盖。

基本路径测试方法适用于模块的详细设计和源程序,下面以选择排序程序 SelectSort 为例,说明具体测试用例设计过程:

```
void selectSort(int V[],int n)  {
    for(int i=0;i<n-1;i++){
        int k=i;
        for(int j=i+1;j<n;j++)
          if(V[j]<V[k])   k=j;
    if(k!=i)   { int work=V[i];V[i]=V[k];V[k]=work;}
    }
}
```

(1) 以详细设计或源代码作为基础,导出程序的控制流图,即将上述程序转换为如图 4-4 所示的控制流图,用数字标号标识各个控制流。

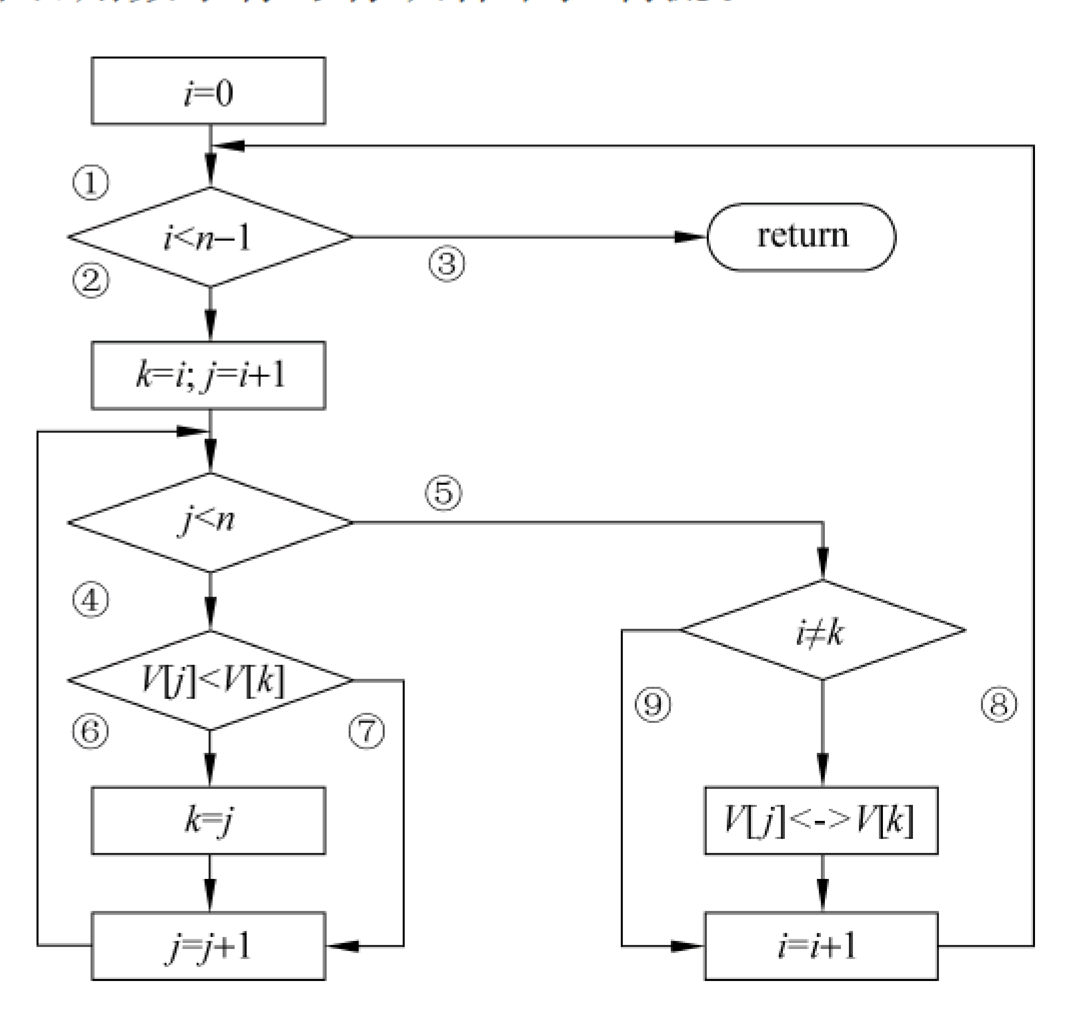


图 4-4 SelectSort 程序的控制流图

(2) 计算得到的控制流图 G 的环路复杂性 $V(G)$。

对控制流图 4-4,可以多种算法求 $V(G)$:

$$V(G)=5(\text{区域数})$$
$$V(G)=14(\text{边数})-11(\text{节点数})=5$$
$$V(G)=4(\text{判定节点数})+1=5$$

(3) 确定线性无关的路径的基本集。根据环路复杂性为 5,可确定该图有 5 条线性无关的基本路径集,分别如下。

Path1: 1—3

Path2：1—2—5—8—…

Path3：1—2—5—9—…

Path4：1—2—4—6—…

Path5：1—2—4—7—…

（4）生成测试用例，确保基本路径集中每条路径的执行。根据判定节点给出的条件选择适当的数据以保证某一条路径可以被测试到。满足上面基本路径集的测试用例如下。

Path1：1—3，取 $n=1$。

Path2：1—2—5—8—3，取 $n=2$；预期结果：路径 5—8—3 不可到达。

Path3：1—2—5—9—3，取 $n=2$；预期结果：路径 5—9—3 不可到达。

Path4：1—2—4—6—5—8—3，取 $n=2$，$V[0]=2$，$V[1]=1$；预期结果：$k=1$，$V[0]=1$，$V[1]=2$。

Path4：1—2—4—6—5—9—3，取 $n=2$，$V[0]=2$，$V[1]=1$；预期结果：$k=1$，路径 9—3 不可到达。

Path5：1—2—4—7—5—8—3，取 $n=2$，$V[0]=2$，$V[1]=1$；预期结果：$k=0$，路径 8—3 不可到达。

Path5：1—2—4—7—5—9—3，取 $n=2$，$V[0]=2$，$V[1]=1$；预期结果：$k=0$，$V[0]=1$，$V[1]=2$。

4.4 数据流测试

4.4.1 定义/使用测试的几个定义

1. 定义节点

节点 n 是变量 v 的定义节点，当且仅当变量 v 的值在节点 n 对应的语句中定义，记作 DEF(v,n)。

2. 使用节点

节点 n 是变量 v 的使用节点，当且仅当变量 v 的值在节点 n 对应的语句中使用，记作 USE(v,n)。

3. 谓词使用

使用节点 USE(v,n)是一个谓词使用，当且仅当 n 是谓词语句，记作 P-use。

4. 定义/使用路径

路径上存在变量 v 的定义节点 DEF(v,m)和使用节点 USE(v,n)，且 m 和 n 是该路径的最初节点和终止节点，则这条路径称为 v 的定义-使用路径，记作 du-path。

5. 定义清除路径

在定义/使用路径中仅存在一个 v 的定义节点，则是定义清除路径，记作 dc-path。

4.4.2 定义/使用路径测试覆盖指标

(1) 全定义准则。
(2) 全使用准则。
(3) 全谓词使用/部分计算使用准则。
(4) 全计算使用/部分谓词使用准则。
(5) 全定义/使用路径准则。

4.5 基于覆盖的测试用例选择

4.5.1 如何使用覆盖率

原则 1：覆盖率不是目的，只是一种手段。因为测试的主要目的还是尽可能地去发现错误。

原则 2：不可能针对所有的覆盖率指标去进行测试，相反，如果只考虑一种覆盖率指标也是不恰当的。

原则 3：不要追求绝对 100%的覆盖率。

4.5.2 使用最少测试用例来达到覆盖

1. 结构化程序的三种基本控制结构

顺序型，构成串行操作；
选择型，构成分支操作；
重复型，构成循环操作。

2. 如何计算最少测试用例数目

一般的程序是由上面三种控制结构嵌套组合而成的，比较复杂，但估算最少测试用例的原则一致。主要采取结构化分解的方法进行，使问题得以简化，如图 4-5 所示。

图 4-5 是表示两个顺序执行的分支结构，分支谓词 P_1 和 P_2 取不同值时，分别执行 a 或 b 及 c 或 d 操作，显然测试该程序，至少要提供 4 个测试用例才能做到路径覆盖，即 ac、ad、bc、bd 操作均能覆盖到。下面看这个最小值 4 的由来：首先可以看出，分支谓词 P_1 引出两个操作，及分支谓词 P_2 引出两个操作，组合起来得到 $2\times2=4$。这里的 2 实际上是由于两个并列的操作 $1+1=2$ 得到的。

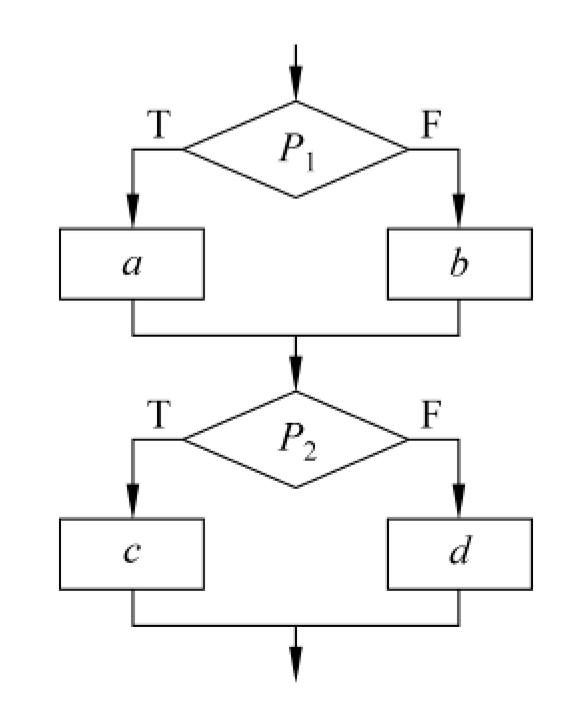


图 4-5　两个串联的判定结构

4.6　本章小结

本章阐述了覆盖率概念、各种逻辑覆盖测试的计算公式、Z 路径覆盖（包括程序控制流图和程序环路复杂性计算的三种方法）、数据流测试的相关定义以及满足路径覆盖的最少测试用例选择方法。

第5章　功能测试

5.1　等价类测试

5.1.1　等价类的概念

所谓等价类，是指某个输入域的子集合，在该子集合中，各个输入数据对于揭露程序中的错误是等效的，故通常假设测试某等价类的代表值就等价于对这一类其他值的测试。等价类有两种：

(1) 有效等价类。是指对程序的规格说明来说是合理的、有意义的输入数据构成的集合，利用它检验程序是否实现了预先规定的功能和性能。

(2) 无效等价类。是指对程序的规格说明来说是不合理的、无意义的输入数据构成的集合，利用它检查程序的功能和性能的实现是否有不符合规格说明的地方。

等价类测试是一种十分实用的黑盒测试方法，设计测试用例通常要经过划分等价类和选取测试用例两步。

5.1.2　等价类测试的类型

(1) 弱一般等价类测试——是指测试用例的设计是通过从每个等价类(区间)选择一个值来实现的。所谓“弱”，是指从各个等价类中选择值时只考虑等价类自身。

(2) 强一般等价类测试——是指设计测试用例时需要考虑等价类之间的相互作用，选取等价类的笛卡儿积的元素值来实现。所谓“强”，是指考虑了等价类之间的相互影响。

(3) 弱健壮性等价类测试——这种测试考虑了从无效等价类取值，即传统的等价类测试。

(4) 强健壮性等价类测试——这种测试也考虑了从无效等价类取值，同时考虑了多个等价类之间的相互影响，从所有等价类笛卡儿积的每个元素中获得测试用例。

5.1.3　等价类测试的原则

(1) 如果输入条件规定取值范围或值的个数，则可确定一个有效等价类和两个无效等价类。

(2) 如果输入条件规定输入值的集合，则可确定一个有效等价类和一个无效等价类。

(3) 如果规定了输入数据的一组值，且程序要对每个输入值分别处理，这时可为每个输入值确立一个有效等价类，此外针对这组值确立一个无效等价类。

(4) 如果规定了输入数据必须遵守的规则，则可确立一个有效等价类和若干个无效

等价类(从不同角度违反规则)。

(5) 若确知已划分的等价类中各元素的处理方式不同,则进一步对等价类进行划分。

5.1.4 等价类方法测试用例设计举例

(1) 以语言标识符规格说明为例,设计这一问题的测试用例。

在某一版本的编程语言中对语言标识符规格作以下规定:“标识符是由以字母开头,后跟字母或数字的任意组合构成的。编译器能够区分的有效字符数为8个,最大字符数为80个”,并且规定:“标识符必须先声明,后使用”、“在同一声明语句中,标识符至少必须有一个”。

为了用等价类划分的方法得到上述规格说明所规定的要求,本着上述的划分原则,建立输入等价类表格,如表5-1所示。

表5-1 等价类表格

输入条件	有效等价类	无效等价类
标识符个数	1个(1),多个(2)	0个(3)
标识符字符个数	1~8个(4)	0个(5),>8个(6),>80个(7)
标识符组成	字母(8),数字(9)	非字母数字字符(10),保留字(11)
标识符第一个字符	字母(12)	非字母(13)
标识符使用	先说明后使用(14)	未说明已使用(15)

根据上述等价类表格,设计覆盖上述所有等价类的测试用例如下。

有效等价类。

① VAR x,T1234567:REAL; }覆盖(1),(2),(4),(8),(9),(12),(14)等价类

② BEGIN x:=3.414;T1234567:=2.732;…

无效等价类。

① VAR:REAL; }0个标识符,覆盖(3)等价类

② VAR x,:REAL; }标识符0个字符,覆盖(5)等价类

③ VAR T12345678;T12345679:REAL; }标识符多于8个字符,覆盖(6)等价类

④ VAR T12345…;REAL; }标识符多于80个字符,覆盖(7)等价类

⑤ VAR T$:CHAR; }标识符有非法字符,覆盖(10)等价类

⑥ VAR GOTO:INTEGER; }标识符为保留字,覆盖(11)等价类

⑦ VAR 2T:REAL; }标识符以非字母开头,覆盖(13)等价类

⑧ VAR PAR:REAL;

BEGIN…

PAP:=SIN(3.14*0.8)/6; }标识符未声明就使用,覆盖(15)等价类

(2) 以三角形问题为例,设计测试用例。

在描述三角形问题时,可能出现的输出为:非三角形、不等边三角形、等腰三角形和

等边三角形，可以根据这些输出标识得到以下的输出等价类：

$R1 = \{< a,b,c >$：有三条边 a、b 和 c 的等边三角形$\}$

$R2 = \{< a,b,c >$：有三条边 a、b 和 c 的等腰三角形$\}$

$R3 = \{< a,b,c >$：有三条边 a、b 和 c 的不等边三角形$\}$

$R4 = \{< a,b,c >$：三条边 a、b 和 c 不构成三角形$\}$

根据上面的等价类设计弱一般测试用例，如表 5-2 所示。

表 5-2 弱一般等价类测试用例

测试用例	a	b	c	预期输出
WN1	5	5	5	等边三角形，属等价类 $R1$
WN2	2	2	3	等腰三角形，属等价类 $R2$
WN3	3	4	5	不等边三角形，属等价类 $R3$
WN4	4	1	2	非三角形，属等价类 $R4$

由于变量 a、b 和 c 的取值需考虑输入数据的组合问题，则强一般等价类测试用例与弱一般等价类测试用例相同。

考虑到 a、b 和 c 的无效值，设计追加的弱健壮等价类测试用例如表 5-3、表 5-4 所示。

表 5-3 追加的弱健壮等价类测试用例(针对无效等价类)

测试用例	a	b	c	预期输出
WR1	−1	5	5	a 取值不在所允许的取值值域内
WR2	5	−1	5	b 取值不在所允许的取值值域内
WR3	5	5	−1	c 取值不在所允许的取值值域内
WR4	201	5	5	a 取值不在所允许的取值值域内
WR5	5	201	5	b 取值不在所允许的取值值域内
WR6	5	5	201	c 取值不在所允许的取值值域内

表 5-4 追加的强健壮等价类测试用例(不良输入)

测试用例	a	b	c	预期输出
SR1	−1	5	5	a 取值不在所允许的取值值域内
SR2	5	−1	5	b 取值不在所允许的取值值域内
SR3	5	5	−1	c 取值不在所允许的取值值域内
SR4	−1	−1	5	a、b 取值不在所允许的取值值域内
SR5	5	−1	−1	b、c 取值不在所允许的取值值域内
SR6	−1	5	−1	a、c 取值不在所允许的取值值域内
SR7	−1	−1	−1	a、b、c 取值不在所允许的取值值域内

5.2 边界值分析

5.2.1 边界值分析的概念

边界值是指对输入等价类和输出等价类而言，位于其边界的值，或稍高于边界的值及稍低于边界的值这样一些特定情况。

使用边界值测试法设计测试用例时，首先应确定边界情况，通常输入等价类与输出等价类的边界就是应着重测试的边界情况。应当选取正好等于、刚刚大于或刚刚小于边界的值作为测试数据。

5.2.2 选择测试用例的原则

(1) 如果输入条件规定值的范围，则应取刚达到这个范围的边界的值，以及刚刚超越这个范围边界的值作为测试输入数据。

(2) 如果输入条件规定值的个数，则用最大个数、最小个数、比最大个数多1、比最小个数少1的数作为测试数据。

(3) 根据规格说明的每个输出条件，使用前面的原则(1)。

(4) 根据规格说明的每个输出条件，使用前面的原则(2)。

(5) 如果程序规格说明中给出的输入域或输出域是有序集合，则选取集合中的第一个元素和最后一个元素作为测试用例。

(6) 如果程序中使用了内部数据结构，则应当选择这个内部数据结构的边界上的值作为测试用例。

(7) 分析规格说明，找出其他可能的边界条件。

5.2.3 边界值方法测试用例设计举例

通常情况下，软件测试所包含的边界测试有以下几种类型：数字、字符、位置、质量、大小、速度、方位、尺寸、空间等。相应地，以上类型的边界值应该在：最大/最小、首位/末位、上/下、最快/最慢、最高/最低、最短/最长、空/满等情况。

对上述几种边界情况设计测试用例的思路，如表5-5所示。

表5-5 利用边界值作为测试数据的例子

项	边 界 值	测试用例的设计思路
字符	起始－1个字符/结束＋1个字符	假设一个文本输入区域允许1～255个字符，输入第1个和第255个字符作为有效等价类；输入第0个和第256个字符作为无效等价类，这几个数值都属于边界条件值

续表

项	边　界　值	测试用例的设计思路
数值	开始位－1/结束位＋1	例如软件要求数据的输入域需要输入 9 位数据，可以使用最简单的 00000-0000 作为最小和 99999-9999 作为最大值，然后刚好使用小于 9 位和大于 9 位的数值来作为边界条件
空间	小于空余空间一点/大于满空间一点	例如在做软盘的数据存储时，使用比最小剩余磁盘空间大一点的文件作为最大值检验的边界条件

5.3　基于判定表的测试

5.3.1　判定表的概念

判定表(Decision Table)，最适合描述在多个逻辑条件取值的组合所构成的复杂情况下，分别要执行哪些不同的动作。判定表由 4 个部分组成，分别是：

条件桩(Condition Stub)——左上部分，列出各种可能的单个条件。

动作桩(Action Stub)——左下部分，列出可能采取的单个动作。

条件项(Condition Entry)——右上部分，针对各种条件给出多组条件取值的组合。

动作项(Action Entry)——右下部分，指出在条件项的各组取值组合下应采取的动作。

判定表分有限条目判定表和扩展条目判定表两大类，把所有条件都是二元条件的判定表称为有限条目判定表；若条件可以有多个值，则对应的判定表称扩展条目判定表。

使用判定表设计测试用例，可以保证测试的严密性和完备性。

5.3.2　基于判定表的测试用例设计举例

在供应商业务处理中，有一个“检查订货单”的功能：当客户订货款项大于 5000 元时，如果客户拖欠款超过 60 天，向客户发一份拒绝供货备忘录，在客户没有还清货款前不发供货单；如果客户拖欠款没有超过 60 天，可以发供货单。如果客户订货款项没有超过 5000 元，而客户拖欠款超过 60 天，仍可以发供货单但还要发一份催款通知单；如果客户拖欠款没有超过 60 天，可以发供货单。

将上述关系用判定表来表示，如表 5-6 所示。

表 5-6　供应商判定表

	规则 1	规则 2	规则 3	规则 4
c1：订货单金额＞5000 元	T	T	F	F
c2：拖欠货款时间＞60 天	T	F	T	F
a1：发拒绝供货备忘录	Y	N	N	N
a2：发供货单	N	Y	Y	Y
a3：发催款通知单	N	N	Y	N

5.4 基于因果图的测试

5.4.1 因果图的适用范围

前面等价类测试和边界值测试方法都是看重输入条件的，但对于输入条件之间的联系考虑不多。若必须考虑多种条件的组合，相应地产生多个动作的方法来设计测试用例，需要利用因果图测试法。

5.4.2 用因果图生成测试用例

基于因果图方法设计测试用例的基本步骤如下：

(1) 根据规格说明，分析和确定原因和结果，并给每个原因和结果赋予一个标识符。

(2) 分析规格说明描述的语义，找出原因与结果之间、原因与原因之间的对应关系，根据这些关系画出因果图。

(3) 由于语法和环境限制，有些原因和原因之间，结果和结果之间的组合不可能出现，可在因果图上用一些记号表明约束和限制条件。

(4) 把因果图转换成判定表。

(5) 根据判定表的每一列，设计测试用例。

5.4.3 因果图法测试用例设计举例

在因果图中，有4种符号，即恒等、非、或、与，如图5-1所示。

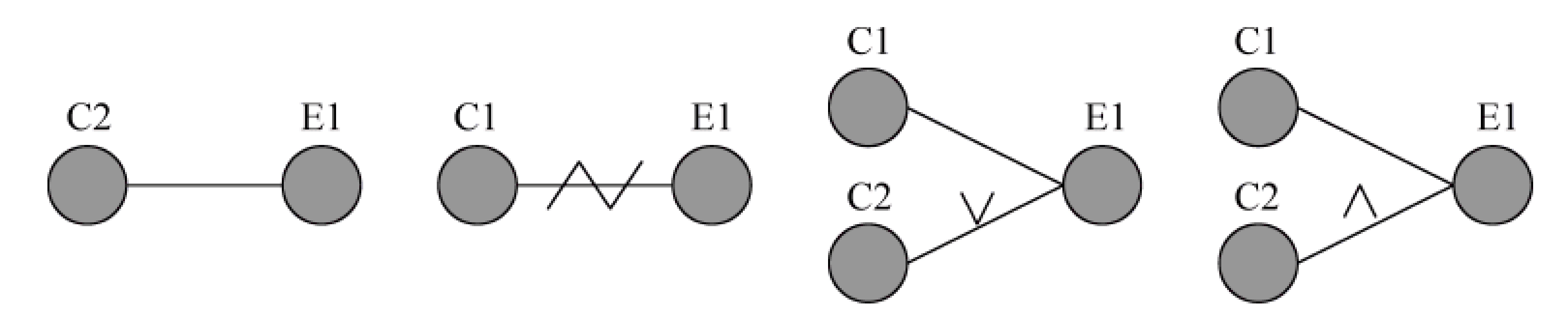


图5-1 因果图的图形符号

在实际问题中，输入状态或输出状态之间还可能存在某些依赖关系，称为“约束”。在因果图中的约束符号主要有5类，如图5-2所示。

用该方法时，要将规格说明分解成可以操作的块，鉴别因和果，制作因果图，注明限制，说明不会出现因/果组合，然后按顺序跟踪因果图中的状态条件，将因果图转换为有限判定表，表中的每列代表一个测试用例，最后将判定表的各列转换为测试用例。

基于因果图方法设计测试用例的基本步骤如下：

(1) 分析具体问题，列出所有的原因和结果，并给每个原因和结果赋予一个标识符。

(2) 进一步分析语义，找出原因与结果之间、原因与原因之间的对应关系，根据这些关系画出因果图。

(a) E(互斥)　(b) I(包含)　(c) O(唯一)　(d) R(要求)　(e) M(屏蔽)

图 5-2　因果图的约束符号

(3) 由于语法和环境限制，有些原因和原因之间，结果和结果之间的组合不可能出现，可在因果图上用一些记号表明约束和限制条件。

(4) 把因果图转换成判定表。

(5) 根据判定表的每一列，设计测试用例。

5.5　基于状态图的测试

5.5.1　功能图及其符号

基于状态图的测试也称为功能图方法，它是用功能图(Function Diagram，FD)形式化的表示程序的功能说明，并机械地生成功能图的测试用例。

功能图模型是由状态图和逻辑功能模型构成的，状态图用于表示输入数据序列以及相应的输出数据，逻辑功能用于表示在状态中输入条件与输出条件之间的对应关系。

一个程序的功能说明通常由动态说明和静态说明组成。动态说明描述了输入数据的次序或迁移的次序，静态说明描述了输入条件和输出条件之间的对应关系。在状态图中，由输入数据和当前状态决定输出数据和后续状态。逻辑功能模型适合于描述静态说明，在逻辑功能模型中，输出数据仅由输入数据决定。

5.5.2　功能图法设计测试用例举例

从状态图生成测试用例的过程如下：

(1) 生成局部测试用例。

(2) 测试路径生成。

(3) 测试用例的合成。

功能图由状态图和布尔函数组成。状态图用如图 5-3 所示的状态和迁移来描述。

下面用状态图表达自动柜员机(ATM)的某规格说明。自动柜员机(ATM)的规格说明如下：

(1) 初始时，ATM 显示“请插入卡片”。

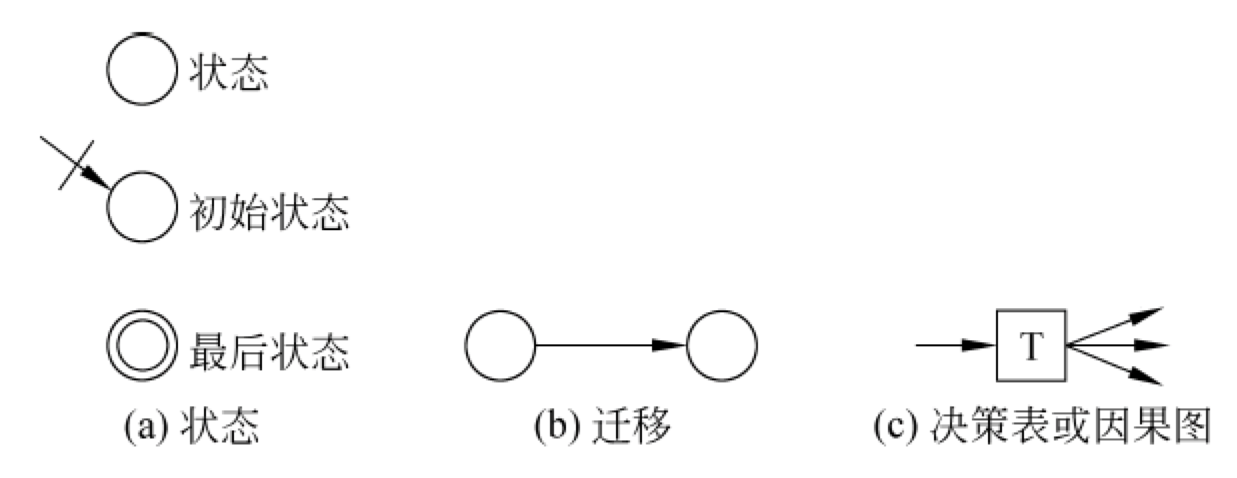


图 5-3　在功能图中的状态迁移符号

(2) 当插入卡片后，ATM 显示“请输入密码”。

(3) ATM 检查输入的密码与文件中保存的密码记录。若相同，则 ATM 显示“请输入金额”；若不同，ATM 检查是否三次都打错了；如是，则 ATM 显示“停止处理”，消去这个记录，重新显示“请插入卡片”；若未达到三次，则 ATM 显示“请输入密码”。

(4) 输入一个取款金额后，ATM 检查它是否小于等于余额，若大于余额，ATM 显示“请输入金额”，等待再次输入金额；否则 ATM 付给要求的现金，报告金额，显示“请插入卡片”。

由上述规格说明，得到自动柜员机(ATM)的状态图如图 5-4 所示。

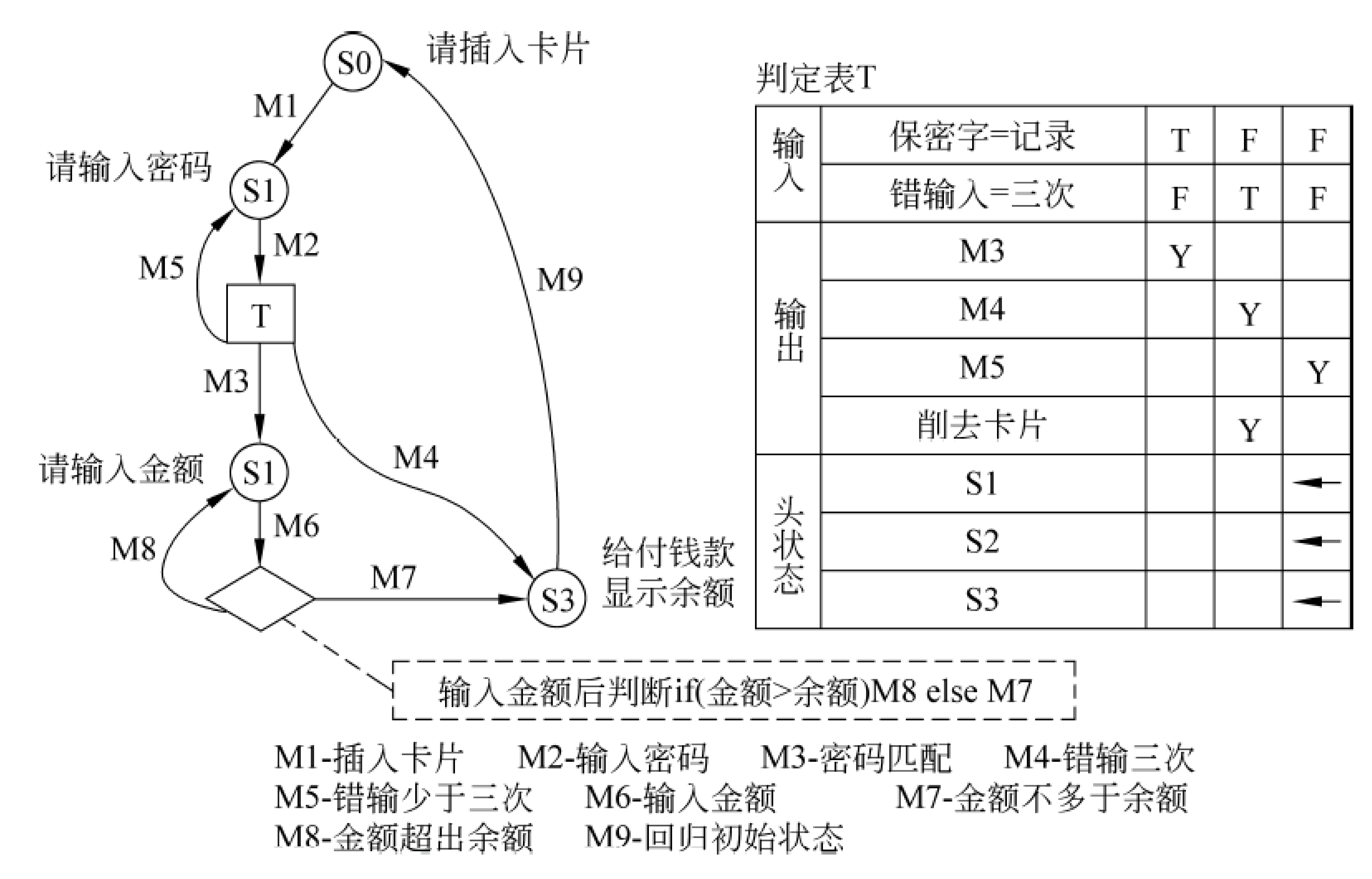


图 5-4　一个状态图的实例

5.6　基于场景的测试

5.6.1　基本流和备选流

基于场景的测试是在面向对象的测试中很有用的设计测试用例的方法。该方法是基于 IBM 公司提出的 Rational 统一过程(RUP)的测试用例生成方法。该方法从系统分析的结果——用例出发，通过对每个用例的场景进行分析，逐步实现测试用例的构造。

场景是指事件触发时的情景，同一事件不同的触发顺序和处理结果就形成了事件流，典型的事件流分为基本事件流（简称基本流）、候补事件流和异常事件流（这两者统称为备选流），如图 5-5 所示。

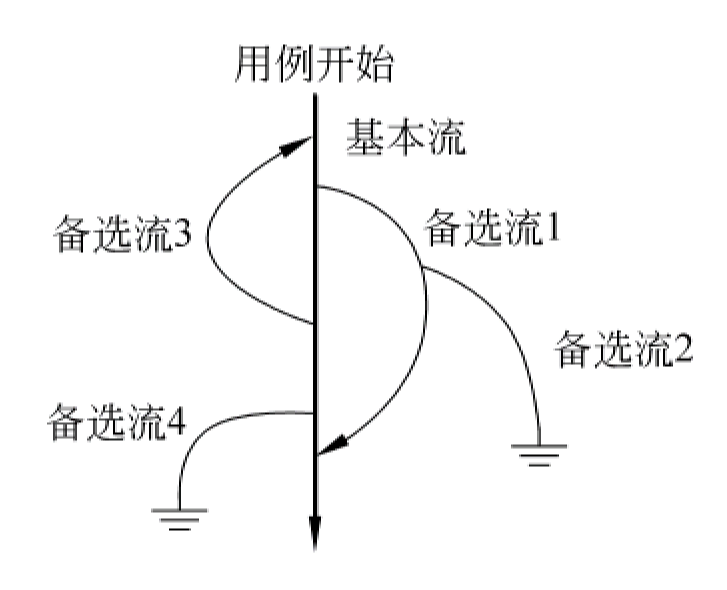


图 5-5　用例的事件流示例

按照图 5-5 中每个经过用例的可能路径，可以确定如表 5-7 所示的不同用例场景。

表 5-7　用例场景与路径的对应关系

场景描述	路　　径	场景描述	路　　径
场景 1	基本流	场景 5	基本流、备选流 3、备选流 1
场景 2	基本流、备选流 1	场景 6	基本流、备选流 3、备选流 1、备选流 2
场景 3	基本流、备选流 1、备选流 2	场景 7	基本流、备选流 4
场景 4	基本流、备选流 3	场景 8	基本流、备选流 3、备选流 4

5.6.2　场景法设计测试用例举例

（1）对用例进行分析。
（2）对用例场景进行分析，发现包含的基本流和备选流。
（3）根据场景设计测试用例。
（4）确定测试数据。

5.7　其他黑盒测试用例设计技术

5.7.1　规范导出法

规范导出的测试是根据相关的规格说明书描述来设计测试用例的，每个测试用例用来测试一个或多个规格说明的陈述语句。规范导出法就是根据陈述所用语句的顺序来相应地为被测单元设计测试用例。

规范导出法是一种正向的测试用例设计技术，其变化形式可以应用到保密分析、安全分析、软件故障分析或其他对单元规格说明做出补充的文件上。

5.7.2 内部边界值测试法

通常可以从单元的功能规格说明中导出等价类和边界值测试，但一个单元内部可能有内部边界，这时只能从单元的结构化规格说明中找到。

内部边界值测试可以用来发现一些内部错误，如误把＜写作＜＝。但内部边界值测试法应作为一种补充方法，在其他方法的最后使用。

5.7.3 错误猜测法

错误猜测是基于经验和其他一些测试技术的。在经验基础上测试设计者猜测错误的类型及发生的位置，并设计测试用例去发现它们。为最大限度地利用有效的经验，建立一个错误类型的列表是一个好方法。

5.7.4 基于接口的测试

基于接口的测试根据模块和它们之间的相互关系特性选择测试数据，具体又可分为：

(1) 输入域测试。

输入域测试的目标是选用域的代表值，从它们的执行中得到整个输入域的测试结果。

(2) 特殊值测试。

特殊值测试是指基于计算功能的特性来选择测试用例的方法，尤其适用于数学计算。

(3) 输出域测试。

通过选择能够使得每个输出域会达到极端值的输入数据作为测试用例来执行测试，就是输出域测试。

5.7.5 基于故障的测试

基于故障的测试(Fault-Based Testing)目标是要证明某个规定的故障不存在。基于测试所涉及范围和广度的不同，基于故障的测试分为：

一种是涉及局部范围的方法，它想要证明一个故障对计算有局部影响，很可能这个影响不会导致程序失败。

另一种是涉及全局范围的方法，它想要证明一个故障会引起一个程序失败。

5.7.6 基于风险的测试

若在测试过程中，首先做一个风险的优先级列表，然后进行测试来探询每个风险，之后随着老风险的消失，新风险的产生，调整测试工作重点到新风险上，这样的测试便是基于风险的测试。

进行风险测试的基本步骤如下：

(1) 决定要分析什么组件或功能。

(2) 确定关心的范围。

(3) 收集想要分析的对象的信息。

(4) 观察每个列表上的每个风险区域，根据手头上的资料确定重要性。

(5) 记录任何不清楚的、影响分析风险能力的事情。

(6) 再次分开检查所有风险。

5.7.7 比较测试

比较测试又称 back-to-back 测试，是对同一软件的不同版本进行测试。就是说，针对同样的需求规格，做出不同的实现，可利用其他黑盒测试技术设计的测试用例作为另一个版本的输入，若每个版本的输出相同就可假定所有的实现都正确，若输出不同就要检查各个版本以发现错误所在。比较测试并不能保证系统中没有错误。如果规格说明本身有错，所有的版本都可能反映该错误。另外，若各个版本产生相同但却错误的结果，比较测试也没有办法发现错误。

5.8 本章小结

本章阐述了等价类测试的实际应用，等价类测试的 4 种类型与具体原则，边界值分析的概念和选择测试用例的 7 个原则，边界值测试用例的设计，判定表的概念和基于判定表的设计测试用例方法，因果图的概念、适用范围及基于因果图的设计测试用例方法，状态图的概念、符号及基于状态图的设计测试用例方法，基本流和备选流的概念及基于场景的设计测试用例方法以及其他黑盒测试用例设计技术，如规范导出法、内部边界值测试法、比较测试等。

第6章　单元测试和集成测试

6.1　单元测试的基本概念

6.1.1　单元测试的定义和目标

单元测试是在软件开发过程中要进行的最低级别的测试活动，或者说是针对软件设计的最小单位——程序模块，进行正确性检验的测试工作，其目标是：

(1) 验证代码是与设计相符合的；

(2) 跟踪需求与设计的实现；

(3) 发现设计和需求中存在的缺陷；

(4) 发现在编码过程中引入的错误。

单元测试的活动模型如图6-1所示。

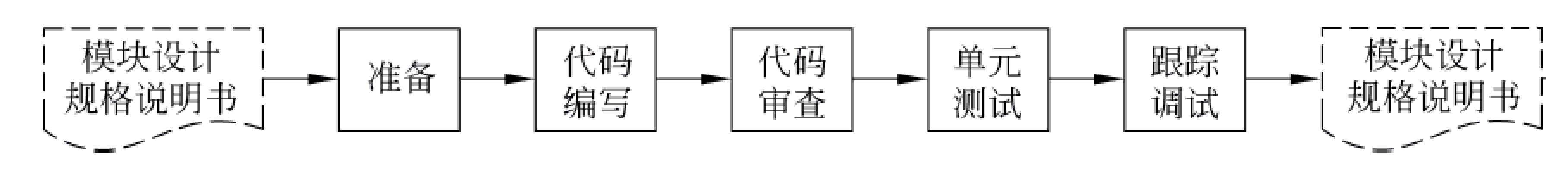


图6-1　单元测试活动模型

6.1.2　单元测试与集成测试、系统测试的区别

1. 单元测试与集成测试的区别

① 测试对象不同。单元测试对象是实现了具体功能的程序单元；而集成测试对象是概要设计规划中的模块及模块间的组合。

② 测试方法不同。单元测试中的主要测试方法是基于代码的白盒测试；而集成测试中主要是使用基于功能的黑盒测试。

③ 测试时间不同。集成测试要晚于单元测试。

④ 测试内容不同。单元测试主要是模块内程序的逻辑、功能、参数传递、变量引用、出错处理及需求和设计中具体要求方面的测试；而集成测试主要验证各个接口、接口之间的数据传递关系，及模块组合后能否达到预期效果。

2. 单元测试与系统测试的区别

① 单元测试属于白盒测试，是从开发者的角度考虑问题，关注的是单元的具体实现、内部逻辑结构和数据流向；系统测试属于黑盒测试，是从用户角度出发看问题，主要目的是证明系统已满足用户的需要。

② 单元测试使问题及早暴露，便于定位解决，属于早期测试；系统测试是一种后期测试，定位错误比较困难。

③ 单元测试允许多个被测单元同时进行测试；系统测试是基于需求规格说明的。

6.1.3 单元测试环境

单元测试的环境并不是系统投入使用后所需的真实环境，而应建立一个满足单元测试要求的环境来做好单元测试工作，环境中要用到一些辅助模块来模拟与被测模块相联系的其他模块（见图 6-2），通常分为两种：

① 驱动模块（Driver），相当于被测模块的主模块。

② 桩模块（Stub），用于代替被测模块调用的子模块。

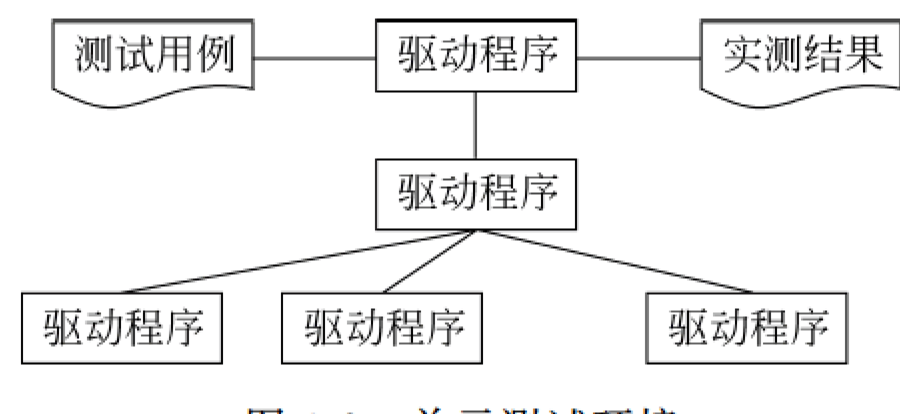


图 6-2 单元测试环境

6.2 单元测试策略

6.2.1 自顶向下的单元测试策略

自顶向下的单元测试策略为：

① 从最顶层开始，把顶层调用的单元用桩模块代替，对顶层模块做单元测试。

② 对第二层测试时，使用上面已测试的单元做驱动模块，并为被测模块编写新的桩模块。

③ 以此类推，直到全部单元测试结束。

这种策略的优点是：可以在集成测试之前为系统提供早期的集成途径。缺点是：随着单元测试的进行，测试过程会变得越来越复杂。因为更改任何一个单元时，就必须重新测试该单元下层调用的所有单元。

6.2.2 自底向上的单元测试策略

自底向上的单元测试的策略为：

① 先对模块调用图上最底层的模块进行测试，使用驱动模块来代替调用它的上层模块。

② 对上一层模块进行单元测试时，用已经测试过的模块做桩模块，并为被测模块编

写新的驱动模块。

③ 以此类推，直到全部单元测试结束。

这种策略的优点是：无须单独设计桩模块；无须依赖结构设计；可为系统提供早期的集成途径。缺点是：随着单元测试的不断进行，测试过程会变得越来越复杂，测试周期延长，测试和维护的成本增加。

6.2.3 孤立测试

这种测试策略是不考虑每个模块与其他模块之间的关系，分别为每个模块单独设计桩模块和驱动模块，逐一完成所有单元模块的测试。

6.2.4 综合测试

在单元测试中，编写桩模块的工作量相当大，故为有效减少开发桩模块的工作量，可以考虑自底向上测试策略与孤立测试策略相结合的综合测试策略。

6.3 单元测试分析

单元测试分析的目的是根据可能的各种情况，确定测试内容，确认这段代码是否在任何情况下都和期望的一致。测试人员要依据详细设计规格说明和源程序清单，理解模块的 I/O 条件和模块的逻辑结构。在进行单元测试分析时，主要从以下几个方面进行考虑。

6.3.1 模块接口

在进行软件测试时，必须输入、输出正确的内容，这样才能使测试有意义。只有数据在模块接口处正确，才能进一步开展工作。

6.3.2 局部数据结构

往往错误的根源在于局部数据结构出错。之所以要对局部数据结构进行检查，主要是为了保证临时存储在模块内的数据在程序执行过程中正确、完整，所以应当认真地设计测试用例。

6.3.3 独立路径

单元测试的一个基本任务是保证模块中每条语句至少执行一次。使用基本路径测试和循环测试有助于发现程序中因计算错误、比较不正确、控制流不适当而造成的错误。

6.3.4 出错处理

一个好的设计应当能够预见各种出错条件，具备适当的出错处理机制，即预设各种出错处理通路。出错处理能力是软件功能的重要组成部分，它保证了软件在运行出错时能够得到及时的补救，保证其逻辑上的正确性。出错处理机制是如此重要，应当对其进行认真的测试。

6.3.5 边界条件

这是最后也是最重要的一项任务。软件时常会在边界上失效，边界测试运用边界值分析技术对边界值及其左右设计测试用例，这可以帮助发现错误。

6.3.6 其他测试分析的指导原则

(1) 验证测试结果的正确性。
(2) 使用反向关联检查。
(3) 交叉检查结果。
(4) 强制一些错误发生。

6.4 单元测试的测试用例设计原则

6.4.1 单元测试的测试用例设计步骤

测试用例设计通常依据的是软件设计文档。对单元测试而言，不仅仅要进行正向测试，还要求进行反向测试。以下6个通用步骤用来指导完成测试用例的设计：
(1) 为系统运行设计用例。
(2) 为正向测试设计用例。
(3) 为逆向测试设计用例。
(4) 为满足特殊需求设计用例。
(5) 为代码覆盖设计用例。
(6) 为覆盖率指标完成设计用例。

6.4.2 单元测试中的白盒测试与黑盒测试

单元测试用例的设计方法通常白盒和黑盒都是可以的，但是以白盒测试为主。

白盒测试应该达到的覆盖率目标是：语句覆盖率达到100%，分支覆盖率达到100%，覆盖程序中的主要路径，即覆盖完成需求和设计功能的代码所在的路径和程序异

常处理执行到的路径。

使用黑盒测试方法设计测试用例通常使用功能覆盖率来量度测试的完整性，而功能覆盖率中最常见的就是需求覆盖，目的是设计一定的测试用例，使得每个需求点都被测试到。其次还包括接口覆盖，目的是通过设计测试用例，使系统的每个接口都被测试到。黑盒测试应达到的覆盖率目标是：程序单元正确实现了需求和设计要求的所有功能；程序单元满足性能要求；程序单元有好的可靠性和安全性。

6.5　集成测试的基本概念

6.5.1　集成测试的定义

所谓集成测试是指根据实际情况对程序模块采用适当的集成测试策略组装起来，对系统的接口以及集成后的功能进行正确性检验。

集成测试又称为组装测试、联合测试、子系统测试或部件测试。

6.5.2　集成测试与系统测试的区别

（1）测试对象不同。集成测试对象是由通过了单元测试的各个模块所集成起来的构件；系统测试对象则除了软件之外，还包括计算机硬件及相关的外设、数据采集和传输机构、支持软件、系统操作人员等整个系统。

（2）测试时间不同。集成测试先于系统测试。

（3）测试方法不同。集成测试通常采用白盒和黑盒相结合的测试方法，也称为灰盒测试；系统测试通常使用黑盒测试。

（4）测试内容不同。集成测试主要是测试各个单元模块之间的接口及各模块集成后的功能；系统测试主要是测试整个系统的功能和性能。

（5）测试目的不同。集成测试的主要目的是发现单元间接口的错误，以及发现集成的软件同概要设计规格说明不一致的地方；系统测试的主要目的是找出软件与系统定义不符合或矛盾的地方。

（6）测试角度不同。集成测试是站在测试人员的角度上进行的；系统测试则站在用户的角度来进行。

6.5.3　集成测试与开发的关系

为了使读者更好地了解集成测试与开发的关系，图6-3给出了软件基本结构图。

软件产品的层次、构件分布、子系统分布为集成测试策略的选取提供了重要的参考依据，从而可以减少集成测试过程中桩模块和驱动模块开发的工作量，促使集成测试快速、高质量地完成。而集成测试可以服务于架构设计，可以检验所设计的软件架构中是否有错误和遗漏，以及是否存在二义性。集成测试和架构设计二者也是相辅相成的关系。

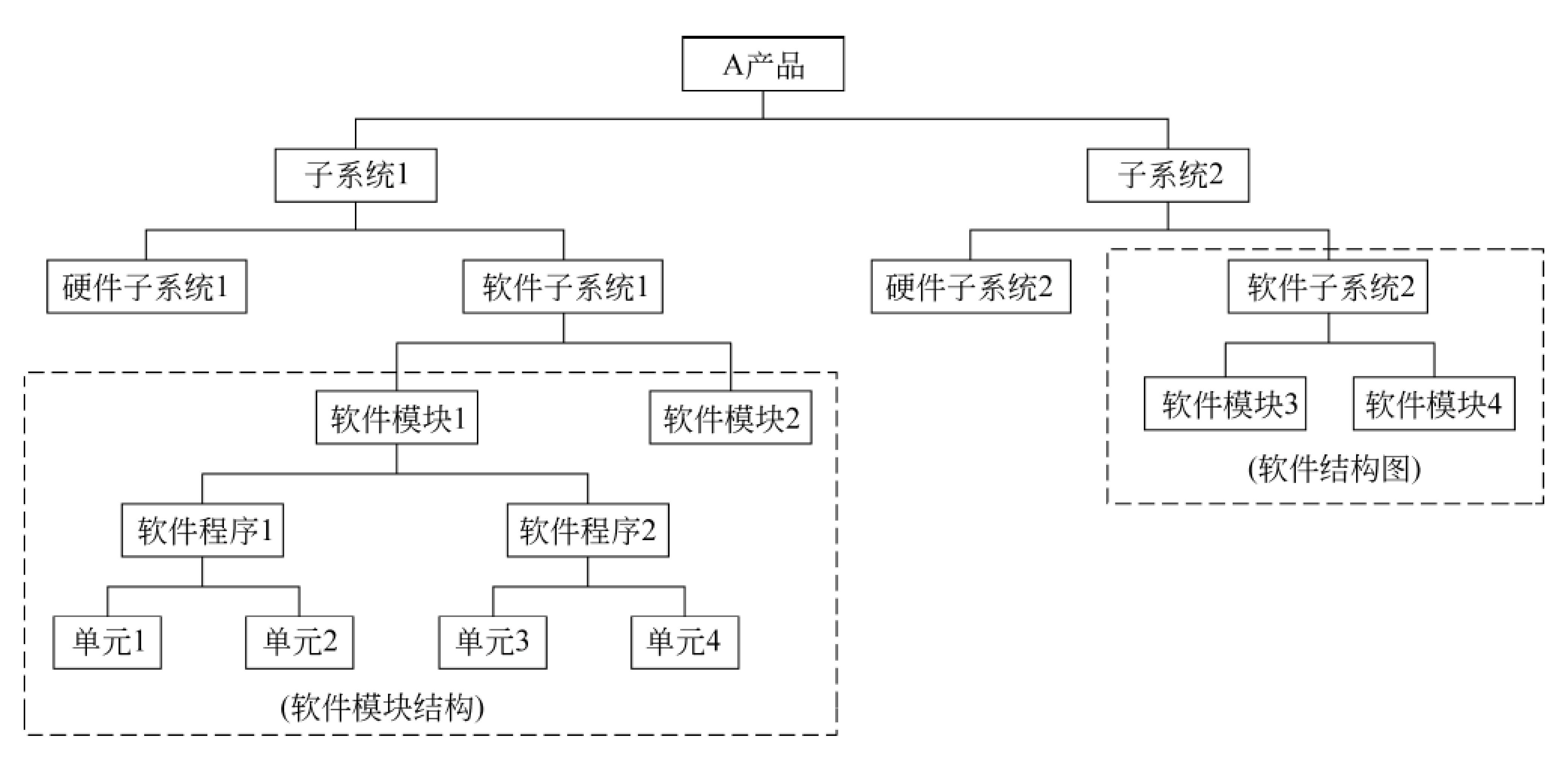


图 6-3　软件基本结构图

6.5.4　集成测试重点

(1) 各个模块连接起来后，穿过模块接口的数据是否会丢失，是否能够按期望值传递给另外一个模块；

(2) 各模块连起来后，需要判断是否仍然存在单元测试时所没发现的资源竞争问题；

(3) 分别通过单元测试的子功能模块集成到一起能否实现所期望的父功能；

(4) 兼容性，检查引入一个模块后，是否对其他与之相关的模块产生负面影响；

(5) 全局数据结构是否正确，是否被不正常地修改；

(6) 集成后，每个模块的误差是否会累计扩大，是否达到了不可接受的程度。

6.5.5　集成测试层次

对于传统软件，集成测试可划分为三个层次：模块内集成测试、子系统内集成测试、子系统间集成测试。

对于面向对象的应用系统，集成测试可分为两个层次：类内集成测试、类间集成测试。

6.5.6　集成测试环境

搭建集成测试环境时，通常从以下几个方面考虑：

(1) 硬件环境；

(2) 操作系统环境；

(3) 数据库环境；

(4) 网络环境；
(5) 测试工具运行环境；
(6) 其他环境。

6.6 集成测试的策略

6.6.1 基于分解的集成策略

(1) 一次性集成方式。

这种方式也称为大突击(Big Bang)测试，是一种非增量式测试。这种方式是把所有系统构件一次性集成到一起进行测试，不考虑构件之间的相互依赖性或可能存在的风险，其主要目标是在最短的时间内把系统组装起来，使用最少的测试来验证整个系统。

(2) 自顶向下的增量式集成方式。

这种集成方式是从顶层控制开始，采用与设计一样的顺序，将模块按系统结构的层次，沿控制层次自顶向下逐步集成测试，以验证系统的接口稳定性。

(3) 自底向上的增量式集成方式。

这种集成方式是从程序模块结构的最底层模块开始集成和测试的，目标是检测整个系统的稳定性。

(4) 混合的增量式(三明治)集成方式。

综合了上面两种集成方式的优点，将系统划分为三层，中间一层为目标层，测试时，对目标层上面使用自顶向下集成策略，对目标层下面使用自底向上集成策略，最后的测试在目标层会合。

(5) 改进的三明治集成。

弥补了三明治集成不能充分测试中间层的缺点，尽可能提高测试的并行性。具体策略是并行目标层、目标层上面一层、目标层下面一层；并行测试目标层与其上面一层的集成和目标层与其下面一层的集成。

6.6.2 基于功能的集成

基于功能的集成是从功能的角度出发，按照功能的关键程度对模块的集成顺序进行组织，目的是采用增量式集成的方法，尽早验证系统的关键功能。

6.6.3 基于调用图的集成

单元调用图是一种有向图，节点表示程序单元，边表示程序调用。基于调用图的集成方式有两种，即成对集成和相邻集成。

涉及的重要等式有：

$$\text{内部结点个数} = \text{结点总数} - (\text{源结点个数} + \text{汇结点个数})$$

邻居个数 = 内部结点个数 + 源结点个数 = 结点总数 − 汇结点个数

6.6.4 基于路径的集成

基于路径的集成测试中，把功能测试和结构测试的方法结合到一起，但基于路径集成的测试需要投入标志 MM-路径的时间。

6.6.5 基于进度的集成

基于进度的集成（Schedule-Based Integration）是在兼顾进度和质量两者之间寻找一个均衡点，该集成的基本策略就是把最早可获得的代码拿来立即进行集成，必要时开发桩模块和驱动模块，在最大限度上保持与开发的并行性，从而缩短项目的集成时间。

6.6.6 基于风险的集成

基于风险的集成（Risk-Based Integration）基于一个假设，即系统风险最高的模块间的集成往往也是错误最集中的地方，因此尽早验证这些接口有助于系统的快速、稳定开发。

6.7 集成测试分析

6.7.1 体系结构分析

首先，跟踪需求分析，对要实现的系统划分出结构层次图。

其次，是对系统各个组件之间的依赖关系进行分析，然后据此确定集成测试的粒度，即集成模块的大小。

6.7.2 模块分析

一般，可从以下几个角度进行模块分析：

（1）确定本次要测试的模块；

（2）找出与该模块相关的所有模块，并且按优先级对这些模块进行排列；

（3）从优先级别最高的相关模块开始，把被测模块与其集成到一起；

（4）然后依次集成其他模块。

6.7.3 接口分析

接口的划分要以概要设计为基础，一般通过以下几个步骤来完成：

(1) 确定系统的边界、子系统的边界和模块的边界。
(2) 确定模块内部的接口。
(3) 确定子系统内模块间的接口。
(4) 确定子系统间的接口。
(5) 确定系统与操作系统的接口。
(6) 确定系统与硬件的接口。
(7) 确定系统与第三方软件的接口。

6.7.4 可测试性分析

系统的可测试性分析应当在项目开始时作为一项需求提出来，并设计到系统中去。在集成测试阶段，分析可测试性主要是为了平衡随着集成范围的增加而导致的可测试性下降。尽可能早地分析接口的可测试性，提前为测试的实现做好准备。

6.7.5 集成测试策略的分析

集成测试策略分析主要根据被测对象选择合适的集成策略。一般来说，一个好的集成测试策略有以下特点：
(1) 能对被测对象进行比较充分的测试。
(2) 能使模块与接口的划分清晰明了。
(3) 要使投入的集成测试资源大致合理。

6.7.6 常见的集成测试故障

大部分集成测试的故障与接口有关，一般的接口错误包括：
(1) 配置/版本控制错误。
(2) 遗漏、重叠或冲突的函数。
(3) 文件或数据库使用不正确或不一致的数据结构。
(4) 文件或数据库使用冲突的数据视图/用法。
(5) 破坏全局存储或数据库数据的完整性。
(6) 由于编码错误或未预料到的运行时绑定导致的错误方法调用。
(7) 客户发送违反服务器前提条件的消息。
(8) 错误的对象和消息的绑定。
(9) 参数错误或不正确的参数值。
(10) 由不正确的内存管理分配/回收引起的失效。
(11) 构件之间的冲突。
(12) 资源竞争，目标环境不能分配象征性装载所需的资源。

6.8 集成测试的测试用例设计

集成测试一般采用的是介于白盒测试和黑盒测试之间的灰盒测试,它综合使用了白盒测试和黑盒测试中的测试分析方法。一般而言,在经过集成测试分析之后,测试用例的大致轮廓已经确定。集成测试用例设计的基本要求是:充分保证测试用例的正确性,保证测试用例无误地完成测试项既定的测试目标,满足相应的测试覆盖率要求。

(1) 为系统运行设计测试用例。

设计一些起码能够保证系统运行的测试用例,这些用例用来测试验证最基本的功能。

可使用的测试分析技术有:等价类测试、边界值测试、基于决策表的测试。

(2) 为正向测试设计测试用例。

正向测试重点是验证集成后的模块是否按照设计实现了预期的功能。

可使用的测试分析技术有:输入域测试、输出域测试、等价类测试、状态迁移测试、规范导出法。

(3) 为逆向测试设计测试用例。

逆向测试主要包括分析被测接口是否完成需求规格未描述的功能;检查规格说明可能出现的遗漏;判断接口定义是否有误;判断可能出现的接口异常错误,包括接口数据本身的错误、接口数据顺序错误等。

可使用的分析技术有:错误猜测法、基于风险的测试、基于故障的测试、边界值测试、特殊值测试、状态迁移测试。

(4) 为满足特殊需求设计测试用例,可使用规范导出法。

(5) 为高覆盖率设计测试用例。

单元测试的覆盖重点主要是路径覆盖、条件覆盖等,而集成测试阶段所关注的覆盖主要为功能覆盖和接口覆盖,可使用功能和接口的覆盖分析。

(6) 测试用例补充。

根据项目的变化,按照需求增加和补充集成测试的用例,保证进行充分的集成测试。

6.9 本章小结

本章阐述了单元测试的基本概念及单元测试与集成测试、系统测试的区别,4 种单元测试策略和它们的优缺点,单元测试分析要考虑的 5 个方面,单元测试的测试用例设计原则,集成测试的基本概念和集成测试与系统测试的区别,集成测试的 7 种策略和它们的优缺点,集成测试分析要考虑的 5 个方面以及掌握集成测试的测试用例设计原则。

第 7 章　系 统 测 试

7.1　系统测试概念

7.1.1　什么是系统测试

系统测试是将已经集成好的软件系统，作为计算机系统的一个元素，与计算机硬件、某些支持软件、数据和人员等其他系统元素结合在一起，在实际运行环境下，对计算机系统进行一系列的集成测试和确认测试。

系统测试目标是：通过与系统的需求规格说明进行比较，检查软件是否存在与系统规格不符合或与之矛盾的地方，从而验证软件系统的功能和性能等满足规格说明所指定的要求。

7.1.2　系统测试与单元测试、集成测试的区别

单元测试是在软件开发过程中要进行的最低级别的测试活动，或者说是针对软件设计的最小单位——程序模块，进行正确性检验的测试工作。集成测试是指根据实际情况对程序模块采用适当的集成测试策略组装起来，对系统的接口以及集成后的功能进行正确性检验。

三者的区别在第 6 章分别进行比较过，这里以系统测试为主体再综合比较一下：

(1) 测试方法不同。系统测试主要是黑盒测试，而单元测试、集成测试主要属于白盒测试或灰盒测试的范畴。

(2) 考察范围不同。单元测试主要测试模块内部接口、数据结构、逻辑、异常处理等对象；集成测试主要测试模块之间的接口和异常；系统测试主要测试整个系统相对于用户的需求。

(3) 评估基准不同。系统测试的评估基准是测试用例对需求规格说明的覆盖率，而单元测试和集成测试的评估主要是代码的覆盖率。

7.1.3　集成测试的组织和分工

系统测试组组长：组织系统测试，负责与管理 IT 设备的人员联系，搭建好系统测试的硬件、软件平台。

测试分析员：负责设计和实现测试脚本和测试用例。

测试员：负责执行测试脚本中记录的测试用例。

7.1.4 系统测试分析

在系统测试的各个环节中，比较关键的还是系统测试用例的设计阶段，在做系统测试分析时，通常从以下几个层次进行分析：

1. 用户层

因为用户层面向的最终使用者是用户，因此用户层的测试主要围绕着用户界面的规范性、友好性、可操作性、系统对用户的支持，以及数据的安全性等方面展开。另外，用户层的测试通常还应注意可维护性测试和安全性测试。

2. 应用层

应用层的测试主要是针对产品工程应用或行业应用的测试。从应用软件系统的角度出发，模拟实际应用环境，对系统的兼容性、可靠性、性能等进行测试。针对整个系统的应用层测试，包含并发性能测试、负载测试、压力测试、强度测试、破坏性测试。

3. 功能层

功能层的测试是要检测系统是否已经实现需求规格说明中定义的功能，以及系统功能之间是否存在类似共享资源访问冲突的情况。

4. 子系统层

子系统层的测试是针对产品内部结构性能的测试。

5. 协议/指标层

协议/指标层的测试是针对系统所支持的协议，进行协议一致性测试和协议互通测试。

7.1.5 系统测试环境

软件测试环境的搭建是软件测试实施的一个重要阶段和环节。在软件开发过程中，创建可复用的软件构件库，对于提高开发质量、减少开发费用、保证开发进度有极重要的辅助作用。同样地，测试人员也可以构建软件测试环境库的方式来实现软件测试环境的复用，节省测试时间。

7.2 系统测试的方法

7.2.1 功能测试

功能测试(Functional Testing)属于黑盒测试，是系统测试中最基本的测试。功能测

试主要根据产品的需求规格说明和测试需求列表，验证产品是否符合需求规格说明。

需求规格说明是功能测试的基本输入，所以在做功能测试前，首先要做的就是对需求规格说明进行分析，明确功能测试的重点。

功能测试用例是功能测试工作的核心，常见的测试用例设计方法有：规范导出法；等价类测试；边界值测试；因果图法；基于判定表的测试；正交实验设计法；基于风险的测试。

7.2.2 协议一致性测试

协议一致性测试(Protocol Conformance Testing)主要用于分布式系统，因为在分布式系统中，很多功能的实现是通过多台计算机相互协作来完成的，这要求计算机之间能相互交换信息，所以需要制定一些规则(协议)，而协议实现者往往会因为理解错误而错误地实现了协议，故要对这些协议进行测试。通常包括以下几种类型的协议测试：

(1) 协议一致性测试；

(2) 协议性能测试；

(3) 协议互操作性测试；

(4) 协议健壮性测试。

协议测试常用的测试用例设计方法有：规范导出法；等价类测试；边界值测试。

7.2.3 性能测试

性能测试(Performance Testing)主要用于实时系统和嵌入式系统，性能测试是指测试软件在集成系统中的运行性能，目标是量度系统的性能和预先定义的目标有多大差距。

一个有用的性能测试是压力测试。典型的压力测试实例是当系统同时接收极大数量的用户和用户请求时，需要测量系统的应对能力。

性能测试要有工具支持，在某种情况下，测试人员必须自己开发专门的接口工具。

性能测试常用的测试用例设计方法有：规范导出法；错误猜测法。

7.2.4 压力测试

压力测试(Stress Testing)又称强度测试，是在各种资源超负荷的情况下观察系统的运行情况的测试。压力测试常用的测试用例设计方法有：规范导出法；边界值测试；错误猜测法。

7.2.5 容量测试

容量测试(Volume Testing)是在系统正常运行的范围内测试并确定系统能够处理的数据容量。容量测试是面向数据的，主要目的就是检测系统可以处理目标内确定的数据容量。容量测试常用的测试用例设计方法有：规范导出法；边界值测试；错误猜测法。

7.2.6 安全性测试

安全性测试(Security Testing)就是要验证系统的保护机制能否抵御入侵者的攻击。

保护测试是安全性测试中一种常见的测试,主要用于测试系统的信息保护机制。

评价安全机制的性能与安全功能本身一样重要,其中安全性的性能主要包括:有效性、生存性、精确性、反应时间、吞吐量。

安全性测试常用的测试用例设计方法有:规范导出法;边界值测试;错误猜测法;基于风险的测试;故障插入技术。

7.2.7 失效恢复测试

失效恢复测试(Recovery Testing)的目标就是验证系统从软件或者硬件失效中恢复的能力。失效恢复测试采用各种人为干预方式使软件出错,造成人为的系统失效,进而检验系统的恢复能力。如果这一恢复需要人为干预,则应考虑平均修复时间是否在限定的范围内。

失效恢复测试常用的测试用例设计方法有:规范导出法;错误猜测法;基于故障的测试;基于风险的测试。

7.2.8 备份测试

备份测试(Backup Testing)是失效恢复测试的补充,目的是验证系统在软件或者硬件失效的事件中备份其数据的能力。

备份测试常用的测试用例设计方法有:规范导出法;错误猜测法;基于故障的测试;基于风险的测试。

7.2.9 GUI 测试

GUI 测试与用户友好性测试和可操作性测试有重复,但 GUI 测试更关注对图形界面的测试。

GUI 测试分为两个部分,一方面是界面实现与界面设计的情况要符合;另一方面是要确认界面能够正确处理事件。

GUI 测试设计测试用例一般要从以下 4 方面考虑:

(1) 划分界面元素,并根据界面的复杂性进行分层。通常把界面划分为三个层次,第一层是界面原子层;第二层是界面组合元素层;第三层是一个完整的窗口。

(2) 在不同的界面层次确定不同的测试策略。

(3) 进行测试数据分析,提取测试用例。

(4) 使用自动化测试工具进行脚本化工作。

GUI测试常用的测试用例设计方法有：规范导出法；等价类测试；边界值测试；因果图法；判断表法；错误猜测法。

7.2.10 健壮性测试

健壮性测试(Robustness Testing)又称容错测试，用于测试系统在出故障时，是否能自动恢复或者忽略故障继续运行。

健壮性测试的一般方法是软件故障插入测试。在软件故障插入测试技术中，需要关注三个方面：目标系统、故障类型和插入故障的方法。

健壮性测试常用的测试用例设计方法有：故障插入测试；变异测试；错误猜测法。

7.2.11 兼容性测试

兼容性测试(Compatibility Testing)就是检验被测的应用系统对其他系统的兼容性。

兼容性测试常用的测试用例设计方法有：规范导出法；错误猜测法。

7.2.12 易用性测试

易用性测试(Usability Testing)与可操作性测试类似，是检测用户在理解和使用系统方面是否方便。易用性测试是面向用户的系统测试，包括对被测系统的系统功能、系统发布、帮助文本和过程等的测试。最好在开发阶段就开始进行。

易用性测试常用的测试用例设计方法有：规范导出法；错误猜测法。

7.2.13 安装测试

安装测试(Installation Testing)的目的就是验证成功安装系统的能力。

安装测试常用的测试用例设计方法有：规范导出法；错误猜测法。

7.2.14 文档测试

文档测试(Documentation Testing)主要是针对系统提交给用户的文档进行验证。文档测试的目标是验证用户文档的正确性并保证操作手册的过程能正常工作。

文档测试的测试用例设计方法采用规范导出法。

7.2.15 在线帮助测试

在线帮助测试(Online Help Testing)用于检验系统的实时在线帮助的可操作性和准确性。

在线帮助测试的测试用例设计方法采用规范导出法。

7.2.16 数据转换测试

数据转换测试(Data Conversion Testing)的目标是验证已存在数据的转换并载入一个新的数据库是否有效。

数据转换测试的测试用例设计方法采用规范导出法。

7.3 系统测试的实施

系统测试开始于已集成软件的确认测试,经过对包括软件在内的系统产品进行的 α 测试、β 测试直至验收测试,目的是保证软件产品能按照合同要求工作,满足用户的要求。

7.3.1 确认测试

确认测试(Validation Testing)又称有效性测试,其任务就是确认软件的有效性,即确认软件的功能和性能及其他特性是否与用户的要求一致。

在这一阶段要做的主要工作是进行功能测试和软件配置复审。

7.3.2 α 测试和 β 测试

α 测试是用户在开发环境下进行的测试,也可以是产品供应商内部的用户在模拟实际操作环境下进行的测试。软件在一个自然设置状态下使用,开发者坐在用户旁边,随时记下错误情况和使用中的问题。这是在受控制环境下进行的测试。

β 测试是由软件的多个用户在一个或多个用户的实际使用环境下进行的测试。这些用户是与产品供应商签订了支持产品预发行合同的外部客户,他们要求使用该产品,并愿意返回有关错误信息给开发者。β 测试通常是在不受控制的环境下进行的测试。与 α 测试不同的是,开发者通常不在测试现场。

7.3.3 验收测试

验收测试(Acceptance Testing)是以用户为主的测试,软件开发人员和质量保证人员也应参加,由用户参加设计测试用例,使用用户界面输入测试数据,并分析测试的输出结果。

7.3.4 回归测试

回归测试(Regression Testing)是在软件变更之后,对软件重新进行的测试,其目标

是检验对软件进行的修改是否正确,保证改动不会带来不可预料的行为或者另外的错误。

7.3.5 系统测试问题总结、分析

对系统测试问题的总结与分析是指问题数与严重级别分布,是评估当前系统质量和测试质量的基础。

科学地界定问题级别,是数据分析和评估的基础和科学化的要求。问题严重级别划分如下:

(1) 致命问题——对应于系统的可用性。

(2) 严重问题——用于分析版本稳定性。

(3) 一般问题——用于评估测试效率。

(4) 提示问题——用于产品的完善性指标。

7.4 如何做好系统测试

系统测试实际上贯穿软件开发的周期,在软件生存周期各个阶段都有系统测试设计和实现的过程,下面是做好系统测试的原则:

(1) 所有的测试都应追溯到用户需求。

(2) 在测试工作真正开始之前,尽早开始测试计划。

(3) Pareto 原则应用于软件测试(Pareto 原则表明测试发现的错误中的 80%很可能起源于 20%的程序模块)。

(4) 系统缺陷应记入文档。

7.5 本章小结

本章阐述了系统测试分析的 5 个层次,分别是:用户层、应用层、功能层、子系统层和协议/指标层,常用的系统测试的方法,系统测试的实施步骤以及系统测试的原则。

第 8 章　面向对象软件的测试

8.1　面向对象软件测试的问题

8.1.1　面向对象的基本特点引起的测试问题

面向对象的三个主要特点：封装、继承和多态，这三个特点为测试带来了很多问题。

1. 封装

在面向对象中，封装包含两方面的含义：一是指信息隐蔽，二是指一组相关的变量和方法被封装在同一个类中。测试时需要考虑：

（1）信息隐蔽对测试执行的影响；

（2）实例状态与类的测试序列。

2. 继承

继承是面向对象中的一个重要机制，它允许子类直接获得父类的属性和方法，从而实现对父类的复用。测试时需要考虑：

（1）继承对测试充分性的影响；

（2）误用对测试的影响。

3. 多态

多态是指对一个类的引用可以与多个类的实现绑定，绑定分为静态绑定和动态绑定。静态绑定是指在编译时刻就完成的绑定，而动态绑定是指在运行时刻完成的绑定。动态绑定对测试的影响首先体现在测试的充分性上。

在 C++ 语言对多态的实现中，只有虚方法才可以进行动态绑定，而普通方法不进行动态绑定。

4. 继承和多态复合

在面向对象中，继承和多态复合在一起可以产生多种变化，这一方面可以帮助程序员设计出许多精巧的代码，也使得因使用不当而引起的错误难以被测试发现。测试时需要考虑：

（1）抽象类对测试执行的影响；

（2）误用对测试的影响。

8.1.2 面向对象程序的测试组织问题

通过执行程序代码完成的测试通常包括单元测试、集成测试和系统测试三个主要方面。对于传统的结构化程序，单元测试是指针对完成单一功能的函数的测试，集成测试是指针对程序中的集成结构的测试，而系统测试是指测试整个应用系统是否满足用户需求。

单元测试的基本要求是被测单元能够被独立地测试。在测试面向对象程序时，由于一个类的各个成员方法通常是相互依赖的，因此很难对一个类中的单个成员方法进行充分的单元测试。面向对象的一个类甚至都不能作为可以被独立测试的单元，主要原因是：

(1) 由于继承的存在，一个类通常依赖于其父类和其他祖先类；

(2) 面向对象程序中经常出现多个类相互依赖，从而导致每个类难以被独立地测试。

集成测试一般是针对程序的集成结构进行的。在面向对象的程序中，许多集成机制在传统结构化程序中很少见，对于这些机制的测试难以直接应用到传统结构化程序的集成测试中。类似地，对于由多个类组成的继承树的测试，传统的集成测试技术也难以适用。

8.2 面向对象软件的测试模型及策略

8.2.1 面向对象软件的测试模型

图 8-1 是一个贯穿面向对象软件开发全过程的测试模型。

该模型把面向对象软件的测试活动分为面向对象分析的测试(OOA)、面向对象设计的测试(OOD Test)、面向对象编程的测试(OOP Test)和面向对象软件(OOS Test)的系统测试，而面向对象编程的测试(OOP)又可分为单元测试(Unit)和集成测试(Integration)。



图 8-1 面向对象的测试模型

8.2.2 面向对象分析的测试

面向对象分析的测试主要包括两个方面：

(1) 检查分析结果是否符合相应的面向对象分析方法的要求。

(2) 检查分析结果是否可以满足软件需求。

在 Coad 和 Yourdon 提出的面向对象分析方法中，分析结果主要包括 5 部分：对象、结构、主题、属性和实例连接、服务和消息连接。

8.2.3 面向对象设计的测试

面向对象设计的测试主要包括三个方面：

(1) 对设计结果本身的审查。

(2) 设计结果与分析结果一致性的审查。

(3) 设计结果对编程的支持。

OOD 与 OOA 的主要区别是：在 OOD 中要考虑与实现相关的内容，而在 OOA 中不需要考虑与实现相关的内容。

8.2.4 面向对象编程的测试

面向对象编程的测试主要包括两个方面：

① 对执行代码进行测试；

② 检查代码风格。

对于有一定规模的程序而言，把整个程序放在一起进行充分的测试非常困难，一般的方法是把程序的各个组成部分分别进行测试。有时候需要把几个类放在一起进行测试，这时候应当重点测试各个类之间的交互。当检查代码风格的时候，首先需要检查代码的风格是否符合要求，其次需要检查代码中是否存在不好的控制结构，不好的控制结构通常是隐患，可能会影响以后的开发和维护。

8.2.5 面向对象程序的单元测试

由于面向对象的程序中可独立被测试的单元通常是一个类族或最小是一个独立的类，面向对象的单元测试可以分为几个层次：

(1) 方法层次的测试；

(2) 类和类族层次的测试。

具体的单元测试技术在本章 8.3 节进行详细描述。

8.2.6 面向对象程序的集成测试

在单元测试的基础上，集成测试的目的是测试系统的各个组成部分放在一起是否能够协调一致。这里将介绍的集成测试策略既包含了常见的集成测试策略，也有针对面向对象的程序或常用于面向对象程序的集成测试策略。另外，由于面向对象的程序中类间的连接方式与结构化软件中函数间的连接方式存在不同，在集成测试时需要对其进行有针对性的处理。

8.2.7 面向对象软件的系统测试

为了反映用户的使用情况，系统测试应该尽可能建立在用户实际使用环境上。在进行系统测试时，通常不关心软件的各个实体的具体实现细节以及实体间的连接细节。因此，系统测试主要是黑盒测试。

8.3 面向对象程序的单元测试

由于面向对象的软件中可独立被测试的单元通常是一个类族或最小是一个独立的类，所以面向对象的单元测试可以分为几个层次。

8.3.1 方法层次的测试

对于一个方法，可以将其看作关于输入参数和所在类的成员变量的一个独立函数，如果该函数的内聚性很高，功能也比较复杂，可以对其单独进行测试。一般只有少数方法需要进行单独测试，这是因为有很多方法与成员变量具有很强的关联性。对单个成员方法的测试类似于传统软件测试中对单个函数的测试，很多测试技术都可以应用到这里来。

这里常用的测试技术有：

(1) 等价类划分测试。根据输入参数把取值域分成若干个等价类。

(2) 组合功能测试：针对那些依据输入参数和成员变量的不同取值组合而选择不同动作的方法。

(3) 递归函数测试：自己调用自己的方法。

(4) 多态消息测试。

8.3.2 类层次的测试

很多成员方法会通过成员变量产生相互依赖的关系，这将导致很难对单个成员方法进行充分的测试。合理的测试是将相互依赖的成员方法放在一起进行测试，这就是所谓的类层次测试。

这里常用的测试技术有：

(1) 不变式边界测试。首先准确定义类的不变式，其次寻找成员方法的调用序列，以违反类的不变式，这些调用序列即可作为测试用例。

(2) 模态类测试：模态类是指对该类所接受的成员方法的调用序列设置一定的限制。这时，需要对类的状态进行建模，确定类的不同状态、每个状态下可以接受的成员方法调用以及状态间的转换关系，从而获得类的状态转换图。根据状态转换图，可以生成调用序列来覆盖状态转换图上的边和路径。每个调用序列可以作为一个测试用例。

(3) 非模态类测试：该类所接受的成员方法的调用序列没有任何限制。可以避免很多因状态引起的麻烦，但整个测试不能以状态图为指导。

8.3.3 类树层次的测试

面向对象中有集成和多态现象，所以对子类的测试通常不能限定在子类中定义的成员变量和成员方法上，还要考虑父类对子类的影响。

这里常用的测试技术有：

(1) 多态服务测试。多态服务测试是为了测试子类中的多态方法的实现是否保持了父类对该方法的规格说明。

(2) 展平测试。将子类自身定义的成员方法和成员变量以及从父类和祖先类继承来的成员方法和成员变量全部放在一起组成一个新类，并对其进行测试。展平后的类的规模可能会相当大，这会给测试带来昂贵的代价，因此需要尽可能地减少不必要的代价。在最复杂的情况下，对子类的测试可能只采用展平测试策略。

8.4 面向对象程序的集成测试

8.4.1 面向对象程序的集成测试策略

在单元测试的基础上，集成测试的目的是测试系统的各个组成部分放在一起是否能够协调一致。在集成测试中，除了需要考虑测试用例生成、测试用例执行、测试结果判断等问题外，选择哪些实体进行集成也是一个需要考虑的问题，这就是所谓的集成测试策略问题。面向对象程序集成测试策略主要有：

(1) 传统的集成测试策略。

主要有大突击集成测试、自底向上集成测试、自顶向下集成测试、夹层式集成测试。

(2) 协作集成。

协作集成就是在集成测试时，针对系统完成的功能，将可以相互协作完成特定功能的类集成在一起进行测试。优点是：编写测试驱动和测试桩的开销小。缺点是：当协作关系复杂时，测试难以充分进行；与传统集成测试相比，协作集成通常不完备。

(3) 基干集成。

在嵌入式系统中，基干集成划分为两部分：内核部分（基干部分）和外围应用部分。该方法的优点是集中了传统集成的优点，并对缺点进行了控制，更加适合大型复杂项目的集成。缺点是：必须对系统的结构和相互依存性进行分析；必须开发桩模块和驱动模块；由于局部采用一次性集成策略导致有些接口可能测试不完整。

(4) 高频集成。

高频集成一般采用冒烟测试的方式，即不预测每个测试用例的预期结果，如果测试中未出现反常情况，就认为通过测试。高频集成有三个主要步骤：①开发人员完成要提供代码的增量构件，同时测试人员完成相关的测试包；②集成测试人员将开发人员新增或修改的构件集中起来形成一个新的集成体；③评价结果。

优点是：高效性、可预测性、并行性、尽早查出错误、易进行错误定位、对桩模块需要不是必需的。

缺点是：若测试包过于简单，可能难以发现问题；开始不能平稳集成；若没有建立适当的标准可能会增加风险。

(5) 基于事件（消息）的集成。

基于事件（消息）的集成就是从验证消息路径的正确性出发，渐增式地把系统集成在

一起，从而验证系统的稳定性。

(6) 基于使用的集成。

基于使用的集成(Use-Based Integration)从分析类之间的依赖关系出发，通过从对其他类依赖最少的类开始集成，逐步扩大到有依赖关系的类，最后集成到整个系统。

(7) 客户机/服务器的集成。

客户机/服务器的集成(Client/Server Integration)不存在独立控制轨迹，每个系统构件都有自己的控制策略。优点是：避免了一次性集成的风险；次序没有大的约束；有利于复用和扩充；支持可控制和可重复的测试。缺点是：测试驱动代码和桩代码的开发成本高。

(8) 分布式集成。

分布式集成(Distributed Services Integration)用于测试松散耦合的同级构件之间交互的稳定性，优缺点与客户机/服务器的集成类似。

8.4.2 针对类间连接的测试

集成策略反映了集成测试中如何选择每轮测试的对象，实际测试中为保证测试充分，常考虑测试类间的连接，常用技术有：

(1) 类关联的多重性测试。

在面向对象中，类间的关联关系存在多重性方面的限制，对多重性的测试是针对类间连接测试的重要方面。此时，测试关注的重点是与连接关系有关的增删改操作。

(2) 受控异常测试。

异常处理是多数面向对象编程语言的重要机制，它允许程序的控制流在出现特殊情况时跳转到特定的位置。由于使用异常处理，异常的抛出和异常的接收可以被放在不同的类中，这实际上是类间隐含的控制依赖关系。在测试时，需要尽可能地覆盖这些隐式的依赖关系。

(3) 往返场景测试。

在面向对象中，许多功能是通过多个类相互协作完成的，往返场景测试就是针对类间协作的一种测试技术。本质上讲，往返场景测试就是把与实现特定场景相联系的代码抽取出来，针对这些代码设计具有百分之百(分支)覆盖率的测试用例集。

(4) 模态机测试。

模态机测试类似于类层次的模层次的模态类测试，只是模态类测试是针对一个类进行的，而模态机测试是针对多个类进行的。

8.5 面向对象软件的系统测试

由于系统测试的主要目标是测试开发出来的软件是否是问题空间的一个合理解，因此对于系统测试而言，面向对象软件与传统结构化软件并没有本质区别。

8.5.1 功能测试

通常采用两种技术进行功能测试：一种是基于大纲的测试，这也是传统软件系统测试经常采用的技术；另一种是基于用例的测试，利用OOA文档中的用例进行的系统测试。

8.5.2 其他系统测试

除功能测试外，完整的系统测试还包括性能测试、兼容性测试、易用性测试和文档测试等，这些与传统结构化软件的系统测试方法相同。

8.6 本章小结

本章阐述了因面向对象的基本特点引起的测试问题，面向对象软件测试模型，面向对象软件的单元测试，面向对象软件的集成测试，针对类间连接的测试策略以及面向对象软件的系统测试。

第 9 章　软件性能测试基础理论

9.1　软件性能定义

软件性能是软件的一种非功能特性，它关注的不是软件是否能够完成特定的功能，而是在完成该功能时展示出来的及时性。由于感受软件性能的主体是人，不同的人对同样的软件有着不同的主观感受，而且不同的人对于软件性能关心的视角也不同。下面来看一下不同的人眼中的软件性能定义。

9.1.1　用户眼中的软件性能

从用户的角度来说，软件性能是软件系统对用户提交请求所响应的时间。通俗地讲，如果用户单击一个提交或者输入一个 URL 地址，随后系统把结果呈现在用户眼前，这个过程所花费的时间即为用户对软件性能的直观印象，如图 9-1 所示。

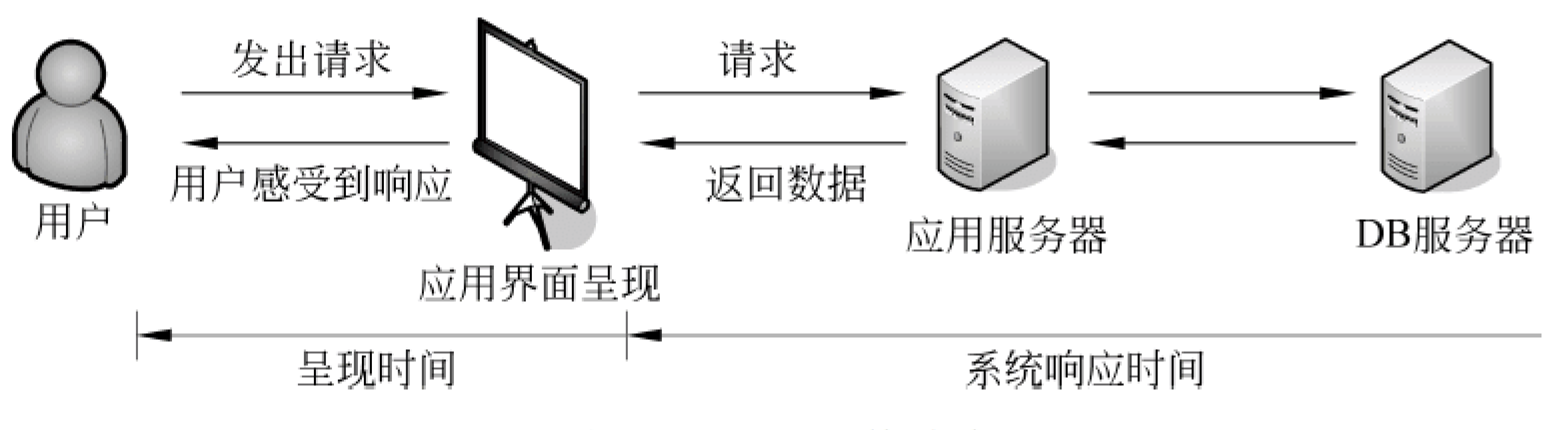


图 9-1　Web 系统响应

用户不关心响应时间中哪些是软件造成的，哪些是硬件造成的。但是用户感受到的响应时间既有客观成分，也有主观成分，甚至是心理因素。

9.1.2　运维人员眼中的软件性能

从运维人员的角度来说，在响应时间方面的理解与用户完全一致。但是运维人员是一群特殊的用户群体，其不仅关注系统的响应时间，还关注服务器系统资源的使用情况。运维人员之所以关注资源的消耗情况，是因为系统响应时间达到要求并不代表系统就能正确地处理客户端提交的请求，如银行系统，假设在处理存款业务时，每笔业务的响应时间为 100ms，但当前的 CPU 和内存的使用率已经达到了 90%，超过正常使用的阈值，这样虽然响应时间符合要求，但是不能保证服务器不出问题，因为服务器已经处于一个临界状态，很可能出现存款不成功的情况，如果这样的话，即使响应时间更短也不能达到性能要求。

另外，运维人员还关注系统硬件资源的可扩展性即规划性能部分。例如，系统现在支

持 100 个用户并发没问题，那么将来支持 200 个用户并发是否会出现性能问题呢？

9.1.3 开发人员眼中的软件性能

从软件开发工程师的角度来说，他们关注用户和管理员关注的所有问题。另外还关注内存泄露、数据库是否出现死锁、中间件以及应用服务器等问题。

9.1.4 Web 前端性能

前端性能，尤其是 Web 前端性能，已经成为目前 Web 应用性能主要被关注的部分之一。随着 Web 应用开发技术的改变，Web 应用响应时间的构成越来越复杂，Ajax 等大量前端技术的应用使得服务器的响应时间在用户感受到的响应时间中所占的比例越来越小。在这种情况下，Web 前端性能越来越受到关注。

Web 应用的前端响应时间指浏览器的页面加载时间。一般而言，浏览器的页面加载时间包括对 HTML 的解析、对页面上的图片及 CSS 等文件的获取和加载、客户端脚本(JavaScript)的执行时间以及对页面进行展现所花费的时间，这部分性能体现就被称为前端性能。与对应的服务器性能不同，前端性能的产生与浏览器的页面元素加载、客户端代码执行以及页面展现相关，与服务器本身并无太大的关系。

9.2 性能测试

9.2.1 性能测试的定义

系统的性能是一个很大的概念，覆盖面非常广泛。对一个软件系统而言，包括执行效率、资源占用、稳定性、安全性、兼容性、可扩展性、可靠性等。性能测试用来保证产品发布后系统的性能满足用户的需求。性能测试在软件质量保证中起重要作用。

性能测试方法是指通过模拟生产环境运行的业务压力量和使用场景组合，测试系统的性能是否满足生产性能要求。这种性能测试的特点有：

这种方法的主要目的是验证系统是否有系统宣称的具体的能力。

这种方法需要了解被测系统的典型场景，并具有确定的性能目标。所谓典型场景就是具有代表性的用户业务操作。其次，这种方法具有确定的性能目标。

这种方法要求在已确定的环境下运行。

9.2.2 性能测试的目标

性能测试的目标是验证软件系统是否能够达到用户提出的性能指标，同时发现软件系统中存在的性能瓶颈、优化软件，最后起到优化系统的目的，主要包括以下几个方面：

(1) 评估系统的能力。测试中得到的负荷和响应时间数据可以被用于验证所计划的

模型的能力，并帮助做出决策。

(2) 识别体系的弱点。受控的负荷可以被增加到一个极端的水平，并突破它，从而修复体系的瓶颈或薄弱的地方。

(3) 系统调优。重复运行测试，验证调整系统的活动得到了预期的结果，从而改进性能。检测软件中的问题，如长时间的测试执行可导致程序发生内存泄露引起的失败，揭示程序中隐含的问题或冲突。

(4) 验证稳定性和可靠性。在一个生产负荷下执行测试一定的时间是评估系统稳定性和可靠性是否满足要求的唯一方法。

9.3 性能测试术语

了解了什么叫软件性能之后，需要对性能测试过程的常用术语有一个详细的了解，为后面的性能测试做准备。下面介绍性能测试过程中的一些常用术语。性能测试过程中的常用术语有：响应时间、并发用户数、吞吐量、吞吐率、TPS、点击率、资源利用率、性能计数器、思考时间等。

9.3.1 响应时间

响应时间是指应用系统从发出请求开始到客户端接收到所有数据所消耗的时间。该定义强调所有数据都已经被呈现到客户端所花费的时间，为什么说是所有数据呢？因为用户体验的响应时间带有主观性，用户认为从提交请求到服务器开始返回数据到客户端的这一段时间为响应时间。

现在对响应时间进行细分，以一个 Web 应用的页面响应时间为例，如图 9-2 所示。从图中可以看到，页面的响应时间可分解为“网络传输时间”($N_1+N_2+N_3+N_4$)和“应用延迟时间”($A_1+A_2+A_3$)，而“应用延迟时间”又可分解为“数据库延迟时间”(A_2)和“应用服务器延迟时间”(A_1+A_3)。

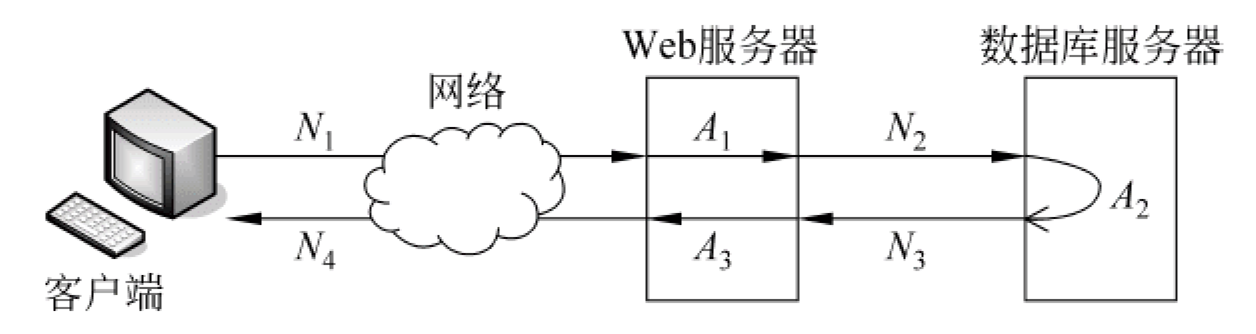


图 9-2 响应时间示例

9.3.2 并发用户数

并发用户数是指同一时刻与服务器进行数据交互的所有用户数量。概念中有两点需要注意。第一：同一时刻，因为并发强调的是用户同时对服务器进行施压。例如：一个人同时挑两件东西，这时表示两件东西同时被这个人挑起来，而如果是先挑一件，再挑另

一件,那么就无法表现出同时的概念,这两件东西也就没有同时施压在这个人身上。第二:强调要与服务器进行数据交互,如果未和服务器进行数据的交互,这样的用户是没给服务器带来压力的。同样是上面的例子,这个人虽然挑了两件东西,但是其中有一件是没有重量的,那就是说只有一件对这个人造成压力。

因此对于并发用户这个概念的理解经常出现以下两种误区:一是认为系统所有的用户都叫做并发用户;二是认为所有的在线用户都是并发用户。在线用户不一定是并发用户,原因是在线用户不一定与系统进行了数据的交互,例如:如果一些在线用户只是查看系统上的一些信息,那么这些在线用户不能作为并发用户计算,因为这些用户并没有与系统进行交互,不会给服务器带来任何压力。

那么并发用户如何计算呢?目前并没有一个精确的计算公式,很多情况下都是根据以往的经验进行估算的。估计行业的不同,并发用户数也会有所不同,像电信行业并发用户数为在线用户的万分之一,如果1000万在线用户,那么需要测试1000个并发用户即可。OA(办公自动化)系统的并发用户数一般是在线用户的2%~20%,所以并发用户数很大程度上是根据经验和行业的一些标准来计算的。

关于并发用户的分布,可以采用以下三种方法进行分析:

(1) 通过参考其他同类产品;

(2) 分析历史数据;

(3) 上线测试运行。

9.3.3 吞吐量

在性能测试过程中,吞吐量是指单位时间内服务器处理客户请求的数量,吞吐量通常使用请求数/秒来衡量,并直接体现服务器的承载能力。

吞吐量作为性能测试过程中主要关注的指标之一,它与虚拟用户间存在一定的联系,当系统没有遇到性能瓶颈时,可以采用下面的这个公式来计算。

$$F = \frac{N_{VU} \times R}{T}$$

其中,F 表示吞吐量,N_{VU} 表示虚拟用户数(Virtual User,VU)的个数;R 表示每个VU发出的请求数量;T 表示性能测试所用的时间。但是如果系统遇到性能瓶颈,这个公式便不再适用。吞吐量与VU之间的关联图如图9-3所示。从图中可以看出,吞吐量在VU增长到一定数量时,软件系统出现性能瓶颈,此时,吞吐量的值并不会随着VU数量的增加而增大,而是趋于平衡。

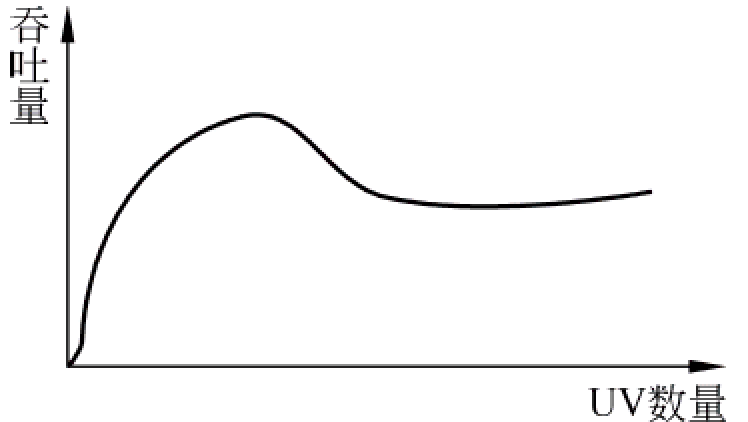


图9-3 吞吐量-VU数量关系图

9.3.4 吞吐率

吞吐率(Throughout)是指单位时间内从服务器返回的字节数,也可以指单位时间内服务器处理客户提交的请求书。它是衡量网络性能的一个重要指标。吞吐率=吞吐量/测试时间。通常情况下,吞吐量的值越大,吞吐率的值也越大,吞吐率的值越大系统的负载能力越强。

9.3.5 TPS

TPS(Transaction Per Second)表示服务器每秒处理的事务数,它是衡量系统处理能力的重要指标。

9.3.6 点击率

点击率(Hit Per Second)是指每秒钟用户向服务器提交的 HTTP 数量。用户每点击一次,服务器端就要对用户提交的请求进行一次处理,从事务的角度来说,如果把每次点击作为一次提交事务来对待,那么点击率与 TPS 的概念是等同的。对于 Web 系统来说,"点击率"是服务器处理的最小单位,点击率的值越大,说明服务器端所能承受的压力越大。因此通常情况下,Web 服务器都具有防刷新的机制,因为客户每刷新一次系统就要响应一次点击,如果不对服务器进行防刷新处理,当用户不停地点击刷新按钮,此时服务器将承受巨大的压力。

需要注意的是,点击一次并不代表客户端只向服务器端发送一个 HTTP 请求,客户每点击一次,都会向服务器端发出多个 HTTP 请求。

9.3.7 资源利用率

资源利用率是指服务器系统中不同硬件资源被使用的程度,资源利用率=资源实际使用量/总的可用资源量,主要包括 CPU 利用率、内存利用率、磁盘利用率、网络等。资源利用率是分析系统性能进而改善性能的主要依据,在配置调优测试的过程中,通过比较配置调优前后系统的资源利用率来判断调优的效果。

9.3.8 性能计数器

性能计数器(Counter)是描述服务器或者操作系统性能的一些数据指标,主要通过添加计数器来观察系统资源的使用情况。性能计数器包括操作系统性能计数器、数据库计数器、应用服务器计数器等。

计数器在性能测试过程中发挥着"监控和分析"的关键作用,尤其是在分析系统的可

扩展性和对性能瓶颈进行定位的时候，计数器的阈值起着非常重要的作用。必须注意的是，一般情况下，单一的性能计数器只能体现系统性能的某一个方面，在性能测试过程中分析测试结果时，必须基于多个不同的计数器进行分析。

在性能测试中常用资源利用率进行横向对比。如在进行性能测试时发现，某个资源的使用率很高，几乎达到100%，假设该资源是CPU，而其他资源的使用率又比较低，这时可以很清楚地知道，CPU是系统的瓶颈。

9.3.9 思考时间

思考时间(Think Time)，也称为"休眠时间"，是指用户在进行操作时，每个请求之间的时间间隔。对于交互系统来说，用户不可能持续不断地发出请求，一般情况下，用户在向服务器端发送一个请求之后，会等待一段时间再发送下一个请求。

在测试脚本中，思考时间为脚本中两条请求语句之间的时间间隔。当前对于不同的性能测试工具提供不同的函数来实现思考时间，在实际的测试过程中，如何设置思考时间是性能测试工程师要关心的问题。

9.4 软件性能测试方法论

"没有规矩，不成方圆。"对性能测试来说，如果没有合适的方法论指导，性能测试很容易成为一种随意的测试行为，而随意进行的性能测试很难取得实际的作用和预期的效果，因此本小节介绍几种常见的性能测试过程和方法。

9.4.1 SEI负载测试计划过程

SEI负载测试计划过程(SEI Load Testing Planning Process)是一个关注于负载测试计划的方法，其目标是产生"清晰、易理解、可验证的负载测试计划"。SEI负载测试计划过程包括6个关注的区域：目标、用户、用例、生产环境、测试环境和测试场景，其主要重点关注以下几个部分。

(1) 生产环境与测试环境的不同：由于负载测试环境与实际的生产环境存在一定的差异，因此，在测试环境上对应用系统进行的负载测试结果很可能不能准确反映该应用系统在生产环境上的实际性能表现，为了规避这个风险，必须仔细设计测试环境。

(2) 用户分析：用户是对被测应用系统性能表现最关注和受影响最大的对象，因此，必须通过对用户行为进行分析，依据用户行为模型建立用例和场景。

(3) 用例：用例是用户使用某种顺序和操作方式对业务过程进行实现的过程。对负载测试来说，用例的作用主要在于分析和分解出关键的业务，判断每个业务发生的频度、业务出现性能问题的风险等。

9.4.2 RBI 方法

RBI(Rapid Bottleneck Identify)方法是 Empirix 公司提出的一种用于快速识别系统性能瓶颈的方法，该方法基于以下一些事实：

(1) 发现的 80%系统的性能瓶颈都由吞吐量制约；

(2) 并发用户数和吞吐量瓶颈之间存在一定的关联；

(3) 采用吞吐量测试可以更快速地定位问题。

RBI 方法首先访问服务器上的“小页面”和“简单应用”，从应用服务器、网络等基础的层次上了解系统吞吐量表现；其次选择不同的场景，设定不同的并发用户数，使其吞吐量保持基本一致的增长趋势，通过不断增加并发用户数和吞吐量，观察系统的性能表现。

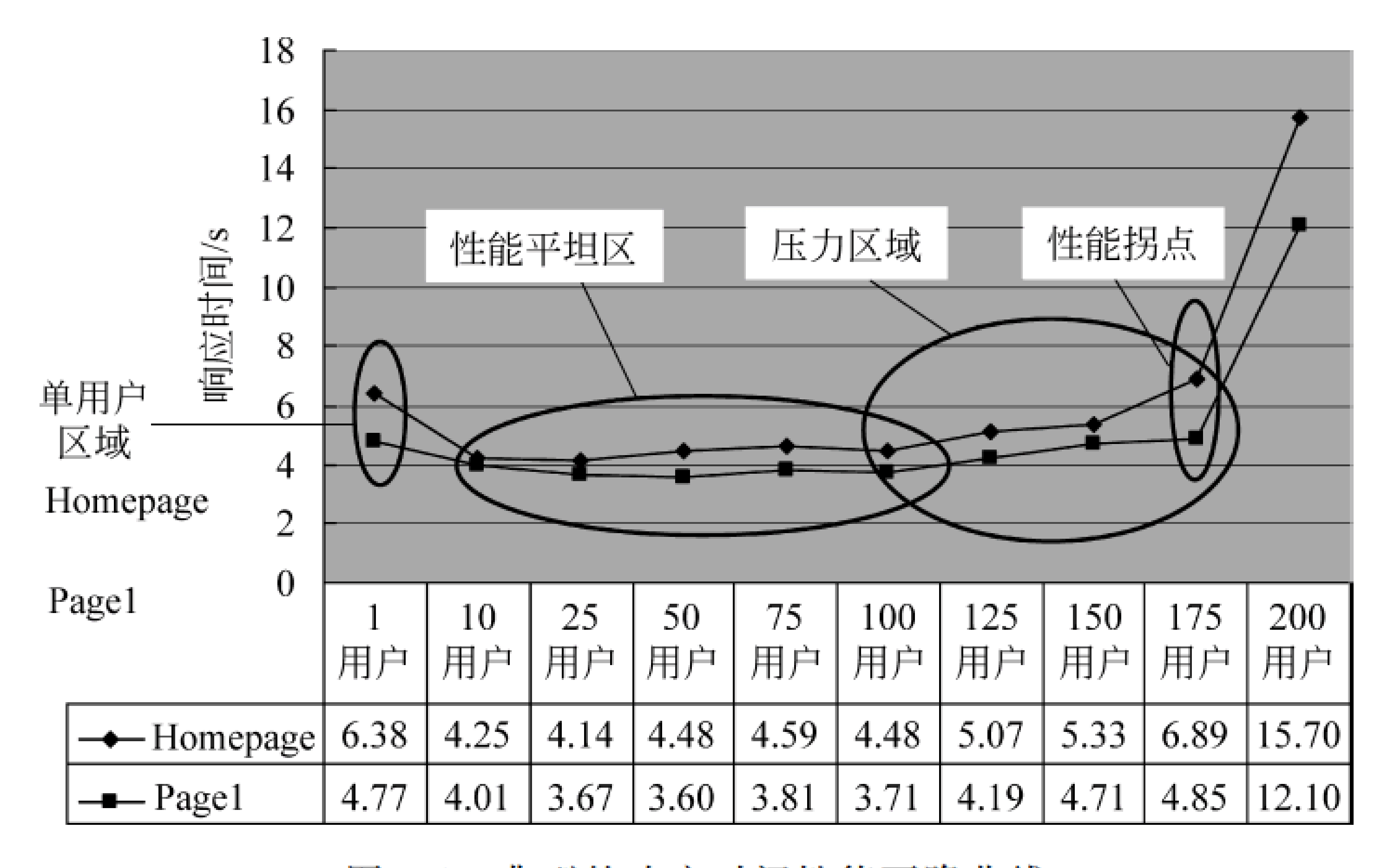
在定位具体的性能瓶颈时，RBI 将性能瓶颈的定位按照一种“自上而下”的分析方式进行分析，首先确定是由并发还是由吞吐量引发的性能表现限制，然后从网络、数据库、应用服务器和代码本身 4 个环节确定系统性能具体的瓶颈。

RBI 方法在性能瓶颈的定位过程中能发生良好的作用，其对性能分析和瓶颈定位的方法值得借鉴，但也不是完整的性能测试过程。

9.4.3 性能下降曲线分析法

性能下降曲线实际上描述的是性能随用户数增加而出现下降趋势的曲线。而这里所说的性能可以是响应时间，也可以是吞吐量或单击数/秒的数据。当然，一般来说，性能主要是指响应时间。

如图 9-4 所示为一条响应时间性能下降曲线。



	1 用户	10 用户	25 用户	50 用户	75 用户	100 用户	125 用户	150 用户	175 用户	200 用户
Homepage	6.38	4.25	4.14	4.48	4.59	4.48	5.07	5.33	6.89	15.70
Page1	4.77	4.01	3.67	3.60	3.81	3.71	4.19	4.71	4.85	12.10

图 9-4　典型的响应时间性能下降曲线

从图 9-4 可以看到，一条响应时间性能下降曲线可以分为以下几个部分：

单用户区域——对系统的一个单用户的响应时间。这对建立性能的参考数值很有

帮助。

性能平坦区——在不进行更多性能调优的情况下所能期望达到的最佳性能。该区域可被用作基线或是 benchmark。

压力区域——应用轻微下降的区域。典型的、最大的建议用户负载是压力区域的开始。

拐点——性能开始急剧下降的点。

这几个区域实际上明确标识了系统性能最优秀的区间、系统性能开始变坏的区间，以及性能出现急剧下降的点。对性能测试来说，找到这些区间和拐点，也就可以找到性能瓶颈产生的地方。

对性能下降曲线分析法来说，主要关注的是性能下降曲线上的各个区间和相应的拐点，通过识别不同的区间和拐点，从而为性能瓶颈识别和性能调优提供依据。

9.4.4 LoadRunner 的性能测试过程

图 9-5 给出了 LoadRunner 的性能测试过程。LoadRunner 将性能测试过程分为计划测试、测试设计、创建 VU 脚本、创建测试场景、运行测试场景和分析结果 6 个步骤。



图 9-5 LoadRunner 的性能测试过程

计划测试阶段主要进行测试需求的收集、典型场景的确定；测试涉及阶段主要进行测试用例的设计；创建 VU 脚本阶段主要根据设计的用例创建脚本；创建测试场景阶段主要进行测试场景的设计和设置，包括监控指标的设定；运行测试场景阶段主要对已创建的测试场景进行执行，收集相关数据；分析结果阶段主要进行结果的分析和报告工作。

LoadRunner 的性能测试过程涵盖了性能测试工作的大部分内容，但由于该过程过于紧密地与 LoadRunner 工具继承，没有兼顾使用其他工具或用户自行设计工具的需求，也不能称为一个普适性的测试过程。

另外，LoadRunner 提供的性能测试过程并未对计划测试阶段、测试设计阶段的具体行为、方法和目的进行详细描述，因此该方法最多只能被称为"使用 LoadRunner 进行测试的过程"，而不是一个适应性广泛的性能测试过程。

9.4.5 Segue 提供的性能测试过程

图 9-6 给出了 Segue 公司 Silk Performer 提供的性能测试过程。该性能测试过程和 Performance Testing Lifecycle 一致，是一个不断 try-check 的过程。

Silk Performance 提供的性能测试过程从确定性能基线开始的，通过单用户对应用的访问获取性能取值的基线，然后设定可接受的性能目标，用不同的并发用户数等重复进

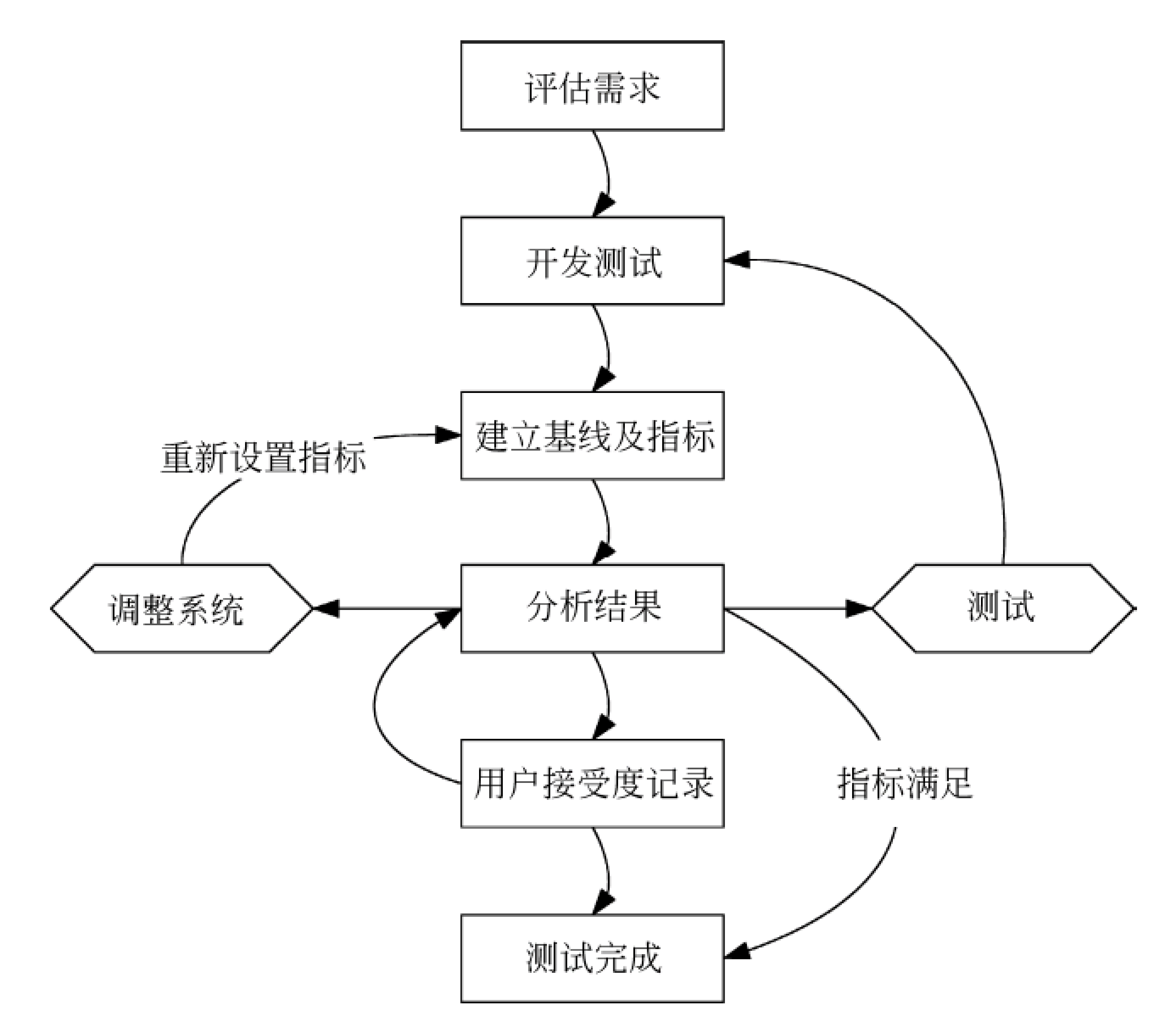


图 9-6 Segue 提供的性能测试过程

行测试。该性能测试方法非常适合性能调优和优化，通过不断重复的 try-check 过程，可以逐一找到可能导致性能瓶颈的地方并对其进行优化。

但 Segue 提供的性能测试过程的模型存在与 LoadRunner 的性能测试过程同样的问题，就是过于依赖工具自身，另外，该过程模型缺乏对计划、设计阶段的明确划分，也没有给出具体的活动和目标。

9.4.6 敏捷性能测试

敏捷性能测试是随着敏捷技术的发展而出现的一种新的性能测试方法。敏捷开发方法将开发定义为短时间段内的迭代，在每个迭代周期中定义迭代目标，通过轻量级的管理方式管理迭代目标的完成。敏捷开发方法在许多项目和组织中得到了广泛的应用，为这些项目和组织带来了巨大的收益，如 Google、Yahoo 等大型互联网都是敏捷开发的忠实用户。

总的来说，敏捷性能测试包括一些特点：

(1) 在每个迭代目标中包含明确的性能目标；

(2) 建立不同层次的性能测试；

(3) 完全或接近完全自动化的性能测试；

(4) 使用测试驱动的方法保证性能与优化性能。

9.5 性能测试过程中的常见风险

9.5.1 识别风险

在进行性能测试的过程中，也存在一定的风险，可能造成最终性能测试不能达到预期的效果，主要风险如下：

(1) 测试环境的差异；

(2) 测试数据的不完善；

(3) 测试需求的不完善；

(4) 领导对测试重要性的意识；

(5) 测试人员的素质。

9.5.2 规避风险

既然在软件性能测试的过程中会存在上述风险，那在实践过程中就要尽量去规避它，减少由这些风险带来的损失。下面从两个案例讲解如何规避风险：

案例一 规避测试环境与真实环境的差异带来的风险为例，可以从以下几个方面着手：

(1) 环境对比，判别差异性；

(2) 识别应用的类型；

(3) 测试环境搭建的针对性；

(4) 测试模型的建立。

案例二 规避测试数据不完善带来的风险，可以从以下几个方面着手：

(1) 分析测试数据的差异性；

(2) 预估生产环境一年内的增长比例；

(3) 测试基础数据的创建。

9.6 本章小结

本章介绍了软件性能测试的基本概念，给出了软件性能的定义，并从三个角度介绍了软件性能的具体体现。此外，还对软件性能的常见术语进行了介绍，并详细探讨了其定义。其次，本章还列出了目前主流的软件性能测试方法。最后，简要介绍了在软件性能测试中存在的风险以及规避风险的方法。

第 10 章　性能测试的应用领域

本章将引入一个“性能测试的应用领域”的概念，通过该概念将性能测试应用的不同场合划分为 5 种不同的性能测试应用领域，然后结合在 2.1 节对各种性能测试方法的讨论，给出一些在不同的性能测试应用领域内进行性能测试时采用的方法的建议。

本书中采用的是一种大范围的性能测试的概念，根据这个概念的界定，性能测试包括以下方法：验收性能测试、负载测试、压力测试、配置测试、并发测试、可靠性测试。

10.1　性能测试的方法分类

性能测试的方法比较多，负载测试和压力测试是比较常见的类型。此外，并发测试等也会在性能测试中进行讨论。

10.1.1　验收性能测试

验收性能测试（Acceptance Performance Testing）方法通过模拟生产运行的业务压力量和使用场景结合，测试系统的性能时能满足生产性能要求。这是一种最常见的方法，通俗地说，这种测试方法就是要在特定的运行条件下验证系统的能力情况。

验收性能测试具有以下特点：

（1）这种方法的主要目的是验证是否具有系统宣称具有的能力。验收性能测试方法包括确定用户场景、给出需要关注的性能指标、测试执行、测试分析几个步骤，这是一种完全确定了系统运行环境和测试行为的测试方法，其目的只能是依据事先的性能规划，验证系统是否达到其宣称具有的能力。

（2）这种方法需要事先了解被测试系统的典型场景，并具有确定的性能目标。验证性能测试需要首先了解被测系统的典型场景，所谓典型场景，是指具有代表性的用户业务操作，一个典型场景包括操作序列和并发用户数量条件。其次，这种方法需要有确定的性能目标，性能目标的描述方式一般为：要求系统在 100 个并发用户的并发条件下进行 A 业务操作，响应时间不超过 5s。

（3）这种方法要求在已确定的环境下运行。

验收性能测试方法的运行环境必须是确定的。软件系统的性能表现与很多因素相关，无法根据系统在一个环境上的表现去推断其在另一个不同环境中的表现，因此对这种验证性的测试，必须要求测试时的环境都已确定。

10.1.2 负载测试

负载测试(Load Testing)方法在被测系统上不断增加压力,直到性能指标(如响应时间)超过预定指标或者某种资源使用已经达到饱和状态。

负载测试方法可以找到系统的处理极限,为系统调优提供数据,有时也被称为可置性测试(Scalability Testing)。该方法具有以下特点:

(1) 这种性能测试方法的主要目的是找到系统处理能力的极限。

(2) 负载测试方法通过“检测—加压—性能指标超过预期”的手段,找到系统处理能力的极限,该极限一般会用“在给定条件下最多120个并发用户访问”这样的描述来体现。

(3) 这种性能测试方法需要在给定的测试环境下进行,通常也需要考虑被测系统的业务压力量和典型场景,使得测试结果具有业务上的意义。负载测试方法由于涉及预定的性能指标等需要进行比较的数据,也必须在给定的测试环境下进行。另外,Load Testing方法在“加压”的时候,必须选择典型的场景,在增加压力时保证这种压力具有业务上的意义。

(4) 这种性能测试方法一般用来了解系统的性能容量,或是配合性能调优来使用。

10.1.3 压力测试

压力测试(Stress Testing)方法测试系统在一定的饱和状态下,例如CPU、内存等在饱和使用的情况下,系统能够处理的会话能力,以及系统是否会出现错误。

压力测试方法具有以下特点:

(1) 这种性能测试方法的目的是检查系统处于压力情况下时的应用测性能表现。压力测试方法通过增加访问压力,使应用系统的资源保持在一定的水平,这种测试方法的主要目的是检验此时的应用表现,重点在于有无出错信息产生、系统对应用的响应等。

(2) 这种性能测试一般通过模拟负载等方法,使得系统的资源达到较高的水平。

一般情况下,会把压力设定为“CPU使用率达到75%以上、内存使用率达到70%以上”这样的描述,在这种情况下测试系统响应时间、系统有无产生错误。除了CPU和内存使用率的设定外,JVM的可用内存、数据库的连接数、数据库服务器CPU利用率等都可以作为压力的依据。

(3) 这种性能测试方法一般用于测试系统的稳定性。

用压力测试的方法考察系统的稳定性是出于这样的考虑:如果一个系统能够在压力环境下稳定运行一段时间,那么这个系统在通常的运行条件下应该可以达到令人满意的程度。”在压力测试中,会考虑系统在压力下是否会出现错误,测试中是否有内存的泄露等问题。

10.1.4　配置测试

配置测试(Configuration Testing)方法通过对被测系统软硬件环境的调整,了解各种不同环境对系统性能影响的程度,从而找到系统各项资源的最优分配原则。

配置测试具有以下特点:

(1) 配置测试性能测试方法的主要目的是了解各种不同因素对系统性能的影响程度,从而判断出最值得进行的调优操作。

(2) 此配置测试方法不同于与功能测试并列的那个"配置测试"方法。对整个系统来说,配置测试是针对软件而言的,其主要目的是验证软件能否在不同的软硬件中正常运行,是功能上的验证。而这里提到的配置测试方法,是在性能测试领域内的配置测试方法,其主要目的是了解各种因素对系统性能的影响程序,从而判断出对性能影响最大的因素。

(3) 这种性能测试方法一般在对系统性能状况有初步的了解后进行。

配置测试方法需要在确定的环境、操作步骤和压力条件下进行。该方法在每次执行测试时更换、扩充硬件设备,调整网络环境,调整应用服务器和数据库服务器的参数设置,比较每次的测试结果,从而确定各个因素对系统性能的影响,找出影响最大的因素。

(4) 这种性能测试方法一般用于测试用户性能调优和规划能力。

配置测试方法主要用于性能调优领域,可以实现调优的持续进行。另外,在规划能力领域内,该方法也常被用来评估该如何调整才能实现系统的扩展性。

10.1.5　可靠性测试

可靠性测试(Reliability Testing)方法通过给系统加载一定的业务压力,让系统持续运行一段时间,测试系统在这种条件下是否能稳定运行。

可靠性测试方法具有以下特点:

(1) 这种性能测试方法的主要目的是验证系统是否支持长期稳定地运行。

这种性能测试方法的主要目的是验证系统是否支持长期稳定地运行,其原理已在前面用非常粗糙的方式进行了解释。从直观上讲,在较大的压力下进行一个较长时间的测试,如果系统在测试中不出现问题或是不好的征兆,基本上可以说明系统具备长期稳定运行的条件。

(2) 这种性能测试方法需要在压力下持续一段时间运行。

既然是稳定性测试,至少需要让系统在压力下运行一段时间。这段时间的具体数值需要根据系统的稳定性要求确定。

(3) 测试过程中需要关注系统的运行情况

在运行的过程中,一般需要关注系统的内存使用情况、系统的其他资源使用以及系统响应时间有无明显的变化。如果在测试过程中发现,随着时间的推移,响应时间有明显的变化,或是系统资源使用率有明显的波动,都可能是系统不稳定的征兆。

10.1.6 负载压力测试

(1) 并发性能测试。

并发性能测试(Concurrency Testing)方法通过模拟用户的并发访问目测多用户并发访问同一个应用、同一个模块或者数据记录时是否存在死锁或者其他性能问题。

并发性能测试具有以下特点：

① 这种性能测试方法的主要目的是发现系统中可能隐藏的并发访问时的问题。

② 这种性能测试方法主要关注系统可能存在的并发问题，例如系统中的内存泄露、线程锁和资源争用问题。

③ 这种性能测试方法可以在开发的各个阶段使用，需要相关测试工具的配合和支持。

(2) 疲劳强度测试。

通常采用系统稳定运行情况下能够支持的最大并发用户数或者日常运行用户数，持续执行一段时间业务，通过综合分析交易执行指标和资源监控指标来确定系统处理最大工作量强度性能的过程。疲劳强度测试案例制定的原则是保证系统长期不间断运行的业务量，并且应该尽量去满足该条件。

(3) 大数据量测试。

① 独立的数据量测试。针对某些系统存储、传输、统计、查询等业务进行大数据量测试。

② 综合数据量测试和压力性能测试、负载性能测试、并发性能测试、疲劳性能测试相结合的综合测试方案。

10.2 性能测试应用领域分析

在前面已经提到了“性能测试领域”的概念，并给出了多种不同性能测试方法的说明和解释。本节将详细说明 5 种不同的性能测试应用领域，并将各种不同的性能测试方法与应用领域进行对应。

概括说来，可以将性能测试的应用领域分为 5 个不同领域：

(1) 能力验证；

(2) 规划能力；

(3) 性能调优；

(4) 缺陷发现；

(5) 性能基准比较。

10.2.1 能力验证

能力验证是性能测试最常用的一个领域。一般能力验证采用这样的描述方式：某系统能否在条件 A 下具备 B 技能，重点在于验证系统是否具有某种能力。

能力验证具有以下几个特点：

（1）要求在一个已确定的环境下运行；

（2）需要根据典型场景来设置测试方案与测试用例。

10.2.2 规划能力

规划能力与能力验证有相似之处，但还是存在一些不同的地方，能力验证强调的是在某个条件下具备什么样的能力，而规划能力体现系统如何才能达到要求的性能指标。规划能力问题常常会这样描述：系统如何才能支持未来用户增长的需要，这里强调的是未来能力增长的一个需求，着眼于未来系统的规划。

规划能力领域的特点是：

（1）对系统能力的一种探索性的测试；

（2）可以了解系统的性能及系统性能的可扩展性。

10.2.3 性能调优

性能调优通过测试来调整系统环境，最终使系统能达到最优状态。这是一个持续调优的过程，主要的调优对象有数据参数、应用服务器、系统的硬件资源等。

一个标准性能调优的步骤如下：

（1）确定本次性能测试的基准环境、基准负载和基准性能指标，目的是将这些基准作为后期测试数据的参考对象。

（2）对系统进行调优，再调整系统运行环境和测试方案重复进行性能测试，并记录测试的结果。

（3）将调整后的测试结果与基准数据进行比较，以确定调优的效果，重复执行步骤（2），直到性能指标满足要求。

10.2.4 缺陷发现

性能测试应用领域的主要目标是通过性能测试的手段来发现系统存在的缺陷。很多系统在实验室中没有任何问题，可是当交付用户时就出现了莫名其妙的错误。如果交付给客户后发现多人同时访问速度缓慢或宕机的现象，那么很可能是由于系统性能问题所引起的。

10.2.5 性能基准比较

性能基准比较，顾名思义，就是在不设定明确性能目标的情况下，通过比较得到每次迭代中的性能表现的变化，根据这些变化决定迭代是否达到了预期的目标。

10.3 本章小结

本章主要介绍软件性能测试方法的常用分类，着重介绍了验收性能测试、负载测试、压力测试、配置测试、并发测试、可靠性测试方法及其各自的特点。最后本章从软件的开发、测试和维护阶段可能遇到的实际问题开始，进入了“性能测试应用领域”的概念。本章的“性能测试应用领域”描述将性能测试的应用场景划分为5个不同的领域：能力验证、规划能力、性能调优、发现缺陷以及性能基准比较。

第 11 章　性能测试团队建设

由于软件性能测试与软件功能测试有着显著的不同,也由于其自身的重要性和复杂性,软件性能测试本身具有自己的测试过程模型。相应地,如何组建一个高效的性能测试团队成为有效进行性能测试的关键。

从管理和组织的角度来说,对软件性能测试可以采用项目管理的一般方法和技术来进行管理和组织。但在具体的技术层面上,软件性能测试还是必须要采用对应的软件测试工具,同时也必须服务于软件测试这个目标。因此,可以一般的软件测试模型和项目管理过程为基础,在其中增加部分自动化测试与敏捷开发的模型,以形成适合于软件性能测试需要的测试模型。

本章主要介绍性能测试团队中人员的构成和一些性能测试的过程模型。

11.1　性能测试人员构成

软件性能测试正逐渐成为软件质量保障的一个重要组成部分。而一支合格的性能测试队伍,对于软件性能测试的顺利展开又有着极为重要的意义。

在一个性能测试团队中应该包括这样一些角色:

(1) 项目测试经理角色。

项目测试经理角色负责整个测试项目,对项目的进度负责,其具体的职责包括确定测试目标、制订测试计划、监控计划执行、处理测试项目干系人的交互等。项目测试经理必须具有项目经理的基本技能,能掌握项目的进行。

(2) 测试设计角色。

测试设计角色测试方案和用例,该角色应该具有较强的业务能力,能够根据用户需求和软件需求,从业务的角度分析和整理典型场景,识别出性能需求,并能制订合理可行的测试方案和用例。

(3) 测试开发角色。

测试开发角色负责实现测试设计人员设计的方案和用例,负责测试脚本的编写和维护,确定测试过程中需要监控的性能指标。

(4) 测试执行角色。

测试执行角色按照测试方案和用例,用测试工具组织和执行相应的脚本,监控相关的性能指标,记录测试结果。

(5) 测试分析角色。

测试分析角色需要获得测试执行人员的测试执行结果,对照测试目标分析测试数据和测试过程中获取的性能指标,得出测试结论。根据不同的测试目标,测试分析得出的结论会侧重不同的方面。

（6）支持角色。

支持角色包括系统工程师、网络工程师、数据库工程师。系统工程师主要处理性能测试过程中与环境相关的内容，为测试过程提供支持；网络工程师则保证测试环境中的网络环境，同样，网络工程师也会为测试结果分析提供支持；数据库工程师则保证测试环境中数据库环境的相关内容，并能为测试分析人员提供结果分析上的支持。

表 11-1 给出了个角色的技能和职责描述。

表 11-1 性能测试团队角色的技能和职责描述

角 色	职 责	技 能
测试经理	① 和用户等项目干系人交互，确保测试的外部环境 ② 制订测试计划 ③ 监控测试进度 ④ 发现和处理测试中的风险	① 计划执行和监控能力 ② 风险意识和能力 ③ 外交能力和灵活变通的能力
测试设计	① 定义性能规划 ② 识别用户的性能需求 ③ 建立性能场景	① 业务把握能力 ② 性能需求分析和识别能力
测试开发	① 实现已设计的性能场景 ② 脚本开发、调试 ③ 确定测试时需要监控的性能指标、性能计数器	① 脚本编码和调试能力 ② 理解性能指标和性能计数器
测试执行	① 部署测试环境 ② 执行脚本和场景 ③ 根据监控要求记录测试结果、记录性能指标和性能计数器值	① 搭建测试环境的能力 ② 测试工具使用（执行）的能力 ③ 性能指标和性能计数器值获取和记录的能力
测试分析	① 根据测试结果、性能指标的数值、性能计数器值进行分析 ② 能根据性能规划，分析出系统性能瓶颈，或是给出优化建议	① 掌握性能测试工具的使用方法 ② 掌握应用系统性能领域的相关知识，理解所采用的架构 ③ 熟悉常用的性能分析方法 ④ 具有一定的编码经验
支持角色（系统）	系统支持，协助解决测试工程师无法解决的系统问题	处理系统问题的能力和技能，最好由专职的系统管理员担任这个角色
支持角色（网络）	网络方面的支持，协助测试工程师解决网络方面的问题，在必要时为测试分析角色提供网络方面的分析支持	网络方面的能力和技能，最好由专职的网络管理员担任这个角色
支持角色（数据库）	数据库方面的支持，在必要时为测试分析角色提供数据库方面的支持	数据库方面的能力和技能，最好由专职的 DBA 担任这个角色

11.2 性能测试过程模型

与功能测试相比，性能测试具有更大的复杂性。随着系统的日趋复杂，越来越多性能测试工具的引入，性能测试单单作为系统或验收测试的一个内容来体现，已经不能满足指

导性能测试的需要。

国内性能测试专家提出了一种对性能测试进行管理的方法，该方法基于自动化测试生命周期方法(Automated Test Life-Cycle Methodology，ATLM)和被广泛采用的 TMap 模型，按照 ATLM 的描述方法对性能测试过程进行建模。这就是性能测试过程通用模型(Performance Testing General Model，PTGM)。

PTGM 模型是一个结构化的过程模型，将性能测试过程分为测试前期准备、测试工具引入、测试计划、测试设计与开发、测试执行和管理以及测试分析 6 个步骤，其测试过程模型如图 11-1 所示。

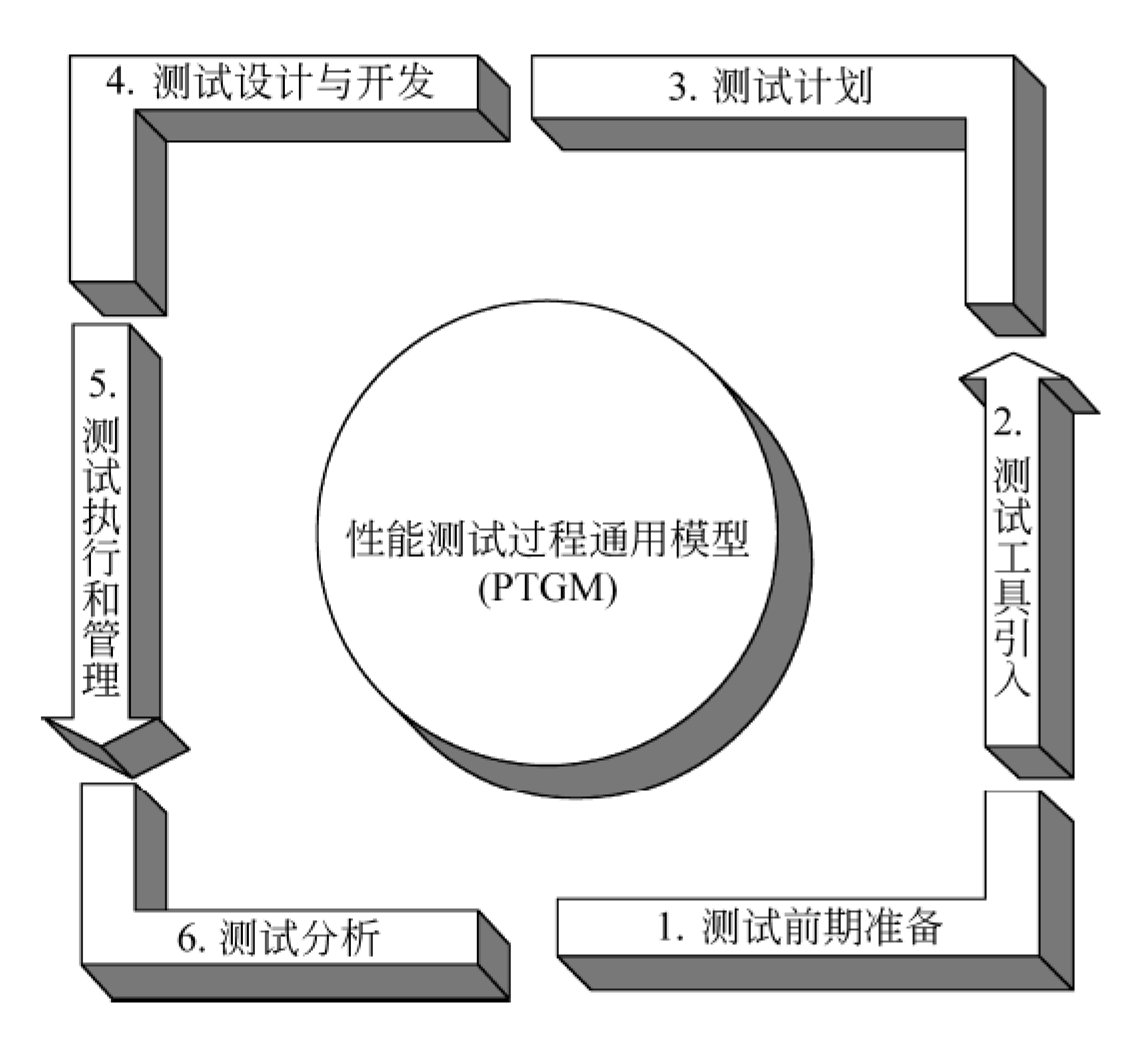


图 11-1　PTGM 模型示意图

考虑到性能测试工作中自动化工具的引入和使用，在 PTGM 模型中引入了一个明确的“测试工具引入”阶段，用以处理和测试工具引入相关的过程；由于性能测试的需求获取与分析、测试团队组建等工作与功能测试存在不同的侧重点(性能测试需求并不像功能测试那么显而易见)，因此在过程模型中增加了一个独立的“测试前期准备”；另外，与功能测试相比，性能测试的测试设计和开发明显会有区别，因此将“测试设计与开发”阶段作为过程中的一个重要阶段。

PTGM 模型的各个阶段都包含了很多活动，下面是 PTGM 模型的详细描述。

11.2.1　测试前期准备

在前期准备阶段，至少要完成两个方面的工作：保证系统稳定和建立合适的测试团队。性能测试一般是软件系统已经开发或是部署完成之后的测试，要求测试对象至少具有一定的稳定性，在功能上基本满足需求。对一个很不稳定或还处于“半成品”状态的软件系统进行测试，没有太大的意义。

具体来说，测试前期包含以下的活动。

（1）系统基础功能验证。

该活动类似于系统测试阶段，每个迭代过程中的 BVT（Build Verification Test），对性能测试而言，这个活动的主要目的是确保当前需要进行性能测试的应用系统已经具备了进行性能测试的条件。

如果性能测试本身属于验收测试的一部分，只需要把性能测试安排在功能验收测试完成之后即可；如果性能测试不在验收测试阶段进行，则必须保证在性能测试之前进行至少一次系统的功能覆盖测试。

（2）组件测试团队。

该活动的目标是建立一个可以进行性能测试的团队。性能测试团队的角色构成在前面的章节已经进行了描述，在测试前期准备阶段，需要根据项目的大致情况，确定人员需要的技能，从而从组织中或是通过招聘挑选合适的人员组成测试组。

（3）测试工具需求确认。

该活动确定测试工具应该具有的功能特性。在这个活动中，需要根据对被测系统的了解和对测试过程的初步规划，给出测试工具应该具备的功能列表，可参见表 11-2。

表 11-2 性能测试工具需求规划表

被测系统环境	测试工具功能需求建议
操作系统环境	测试工具是否能运行在本操作系统上
	测试工具是否支持对本操作系统的监控
应用服务器环境	测试工具能否支持对本应用服务器的监控
数据库环境	测试工具能否支持本数据库的监控
应用使用的协议	本系统使用了哪些协议
	哪些协议需要在性能测试中通过工具进行录制和产生负载
	测试工具能否支持需要进行录制和产生负载的协议
网络环境	是否需要测试工具支持防火墙
	是否需要测试工具支持负载均衡
测试管理支持	测试工具是否能够提供方便的测试结果分析和管理

（4）性能预备测试（可选活动）。

所谓预备测试，是在正式的测试之前，通过简单的探索性测试或是其他方法，对系统的性能表现进行初步的了解。因为这种预备测试是非正式性的，仅仅用来对被测系统的性能建立一个初步印象，所以方法上也比较随意，用人工操作和秒表随机抽查部分操作的性能表现即可。

11.2.2 测试工具引入

性能测试工具在性能测试项目中发挥着不可替代的作用，很难想象一个没有使用任

何性能测试工具而完全依靠手工进行的性能测试。对性能测试来说，为项目测试选择合适的工具，确定测试工具的适用范围，规定和规范测试工具的使用，都不是一件容易的事情。因此把“测试工具引入”作为一个单独的阶段。

测试工具引入阶段包括下列活动。

(1) 工具选择。

性能测试一定会使用自动化测试手段，使用自动测试工具(商业的或是自行开发的)。本活动用于为项目选择合适的工具。

选择的方法是圈定集中可用的工具，对照表 3-2 给出的问题列表，为每个工具进行一个功能符合度的评估，选择符合度最高的工具。如果所有的工具都无法达到所要求的功能符合度，则可以考虑通过创建方式自行构建测试中使用的工具。

(2) 工具应用技能培训。

该活动为项目组的相关参与者进行测试工具的应用技能培训，以使测试活动参与者能够具备测试需要的技能。根据在 3.2.1 节给出的角色和职能，与测试工具相关性最大的是“测试开发”、“测试执行”和“测试分析”角色，因此培训的重点是针对这三种类型的角色进行。

该活动需要达到一定的目标，最好能够在活动开始前确定各种角色人员的详细技能标准，并据此给出培训是否达到预定目标的评判准则。培训活动不一定需要组织内部的人员执行完成，可以通过工具的经销商培训或是外包服务等方式完成。

(3) 确定工具应用过程。

除了工具的应用技能培训外，测试工具引入过程中的另一个重要活动是确定工具的应用过程。

测试工具引入过程中最容易导致的失败就是团队不能达成对测试工具应用范围的一致认可和测试工具应用局限性的一致确认。由于工具经销商常用的夸大和模糊宣传手段，测试工具常常会给不了解的人带来一种“工具无所不能”的印象。如果不能达成这种一致的认识，很容易因测试工具应用的范围发生争执甚至是推诿。

该活动需要确定性能测试工具在测试中的具体应用范围，工具使用过程中的问题解决方法等内容。具体来说，哪些工作使用工具完成？测试工具在使用过程中的问题由谁来解决？测试工具的脚本如何管理？这些问题都应该在这个活动中完成。

11.2.3 测试计划

测试计划阶段用于生成指导整个测试执行的计划。该阶段主要完成测试目标的确定、测试时间的拟定。建议这个阶段的工作分解为以下活动。

(1) 性能测试领域分析。

在性能测试中引入领域的概念可以反映性能测试的直接目的。性能测试的应用领域分为“能力验证”、“规划能力”、“性能调优”和“发现缺陷”4 个领域，在性能测试计划阶段，首先要执行的活动是根据希望本次性能测试达到的目的，分析出性能测试的应用领域。

测试的目的是明确验证系统在固定条件下的性能能力，属于“能力验证”领域，该领域

常见于对特定环境上部署系统的性能验证测试；测试的目的是了解系统性能能力的可扩展性和系统在非特定环境下的性能能力，属于“规划能力”领域，该领域常见于对应用性能可扩展性的测试；测试的目的是通过测试（发现问题）—调优（调整）—测试（验证调优效果）的方法提高系统性能能力，属于“性能调优”领域；测试的目的是通过性能测试手段，发现应用的缺陷，属于“发现缺陷”领域。

确定了性能测试的应用领域之后，可以据此给出性能测试的目标，并可以初步确定可用的性能测试方法。

根据不同的性能测试应用领域分析结果，性能测试的目标定义会有所不同。“性能测试目标”与“性能目标”不同。性能测试目标描述的是性能测试需要达成的目标，而性能目标描述的是性能测试过程中，用于判断性能测试是否通过的标准。

表 11-3 给出了各种不同应用领域的性能测试的性能测试目标和性能目标。

表 11-3　不同应用领域的性能测试目标和性能目标

应用领域	性能测试目标	性 能 目 标
能力验证	验证系统在给定环境中的性能能力	重点关注的关键业务响应时间、吞吐量
规划能力	验证系统的性能扩展能力，找出系统能力扩充的关键点，给出改善其性能扩展能力的建议	业务的性能瓶颈
性能调优	提高系统的性能表现	重点关注的关键业务响应时间、吞吐量
发现缺陷	发现系统中的缺陷	无

（2）用户活动剖析与业务建模。

用户活动剖析与业务建模活动用来寻找用户的关键性能关注点。用户对系统性能的关注往往集中在少数几个业务活动上，在确定性能目标之前，需要先把用户的这些关注点找出来，从而确定最贴近用户要求的性能目标。

用户活动剖析的方法大体分为两种：系统日志分析和用户调查分析。系统日志分析是指通过应用系统的日志了解用户的活动，分析出用户最关注、最常用的业务功能，以及达到业务功能的操作路径；用户调查分析是在不具备系统日志分析条件（例如，该系统尚未交付用户运行实际的业务）时采用的一种估算方法，可以通过用户调查问卷、同类型系统对比的方法获取用户最关注、最常用的业务功能等内容。

经过用户活动分析之后，最终形成的结果类似于以下的描述：

用户最关心的业务之一是 A 业务，该业务具有平均每天 3000 次的业务发生率，业务发生时间集中在 9:00—18:00 的时间段内，业务发生的峰值为每小时 1000 次。A 业务的操作步骤如下所示：

① 用户单击“发布公告”链接。

② 用户在出现的页面中填写公告内容。

③ 用户单击“提交”按钮进行提交。

业务建模是对业务系统的行为及其实现方式和方法的建模，一般采用流程图的方式描绘出各进程之间的交互关系和数据流向。对复杂的业务系统来说，业务建模可以将业务系统清晰地呈现出来，为性能测试提供最直观的指导。

（3）确定性能目标。

性能测试目标根据性能测试需求和用户活动分析结果来确定，确定性能测试目标的一般步骤是首先从需求和设计中分析出性能测试需求，结合用户活动剖析与业务建模的结果，最终确定性能测试的目标。

性能测试需求的来源可以是多方面的，例如，需求文档、用户备忘录或是用户的邮件都可能体现出用户对性能的要求。确定性能目标首先要做的就是从这些文档中获取性能需求。

根据不同的性能测试应用领域分析结果，性能目标定义会稍有不同。

对于“规划能力”领域，性能目标的描述类似如下：

系统的 A 业务在未来的三个月内每天的业务吞吐量达到 4000 笔，找出系统的性能瓶颈并给出可支持这种业务量的建议。

对于“能力验证”领域，性能目标的描述应该类似以下描述：

该应用能够以 1s 的最大响应时间处理 200 个并发用户对业务 A 的访问；峰值时刻有 400 个用户，允许响应时间延长为 3s。

对于“性能调优”领域，最终确定的性能目标的描述类似如下：

通过性能调优测试，本系统的 A 业务和 B 业务在 200 并发用户的条件下，响应时间提高到 3s。

在“能力验证”领域和“性能调优”领域的性能目标描述中，对响应时间、平均的并发用户数量（或是吞吐量）、峰值的并发用户数量（或是吞吐量）、该性能目标针对的业务都进行了明确的定义。当然，在性能测试目标中，还可以加上此时对系统资源使用的定义。一个更为完整的描述类似如下：

该应用能够以 1s 的最大响应时间处理 200 个并发用户对业务 A 的访问，此时服务器的 CPU 占用不超过 75%，内存使用率不超过 70%；峰值时刻有 400 个用户，允许响应时间延长为 3s，此时服务器的 CPU 占用不超过 85%，内存使用率不超过 90%。

（4）制订测试时间计划。

该活动给出性能测试的各个活动起止时间，为性能测试的执行给出时间上的估算。具体方法是根据性能测试活动，为每个活动阶段给出可能的时间估计，最终形成时间上的计划。

11.2.4　测试设计与开发

性能测试的设计与开发阶段包括测试环境设计、测试场景设计、测试用例设计、脚本和辅助工具开发活动。

（1）测试环境设计。

测试环境设计是测试设计中不可缺少的环节。性能测试的结果与测试环境之间的关联性非常大，无论是哪种领域内的性能测试，都必须首先确定测试环境。

对于“能力验证”领域的性能测试来说，测试环境不特定，但也需要设计一个基准的环境。

对于“性能调优”领域的性能测试来说，因为调优过程是一个反复的过程，在每个调优小阶段的末尾，都需要由性能测试来衡量调优的效果，因此必须在开始就给出一个用于衡量的环境标准，并在整个调优过程中，保证每次测试时的环境保持不变。

这里所说的测试环境设计包括系统的软/硬件环境、数据环境设计、环境的维护方法。其中，数据环境设计是非常关键但又最容易被忽视的问题。系统运行在一个已有数万条数据的数据库和一个几乎为空的数据库环境下，其执行查询、插入和删除操作的响应时间显然是不同的。

（2）测试场景设计。

测试场景设计活动用于设计测试活动需要使用的场景。在“确定测试目标”活动中，描述了如何确定测试目标，以及测试目标的一般描述，这个活动需要更详细地将测试目标转化为能够在测试执行中使用的内容。

测试场景模拟的一般是实际业务运行的剖面。对性能测试而言，“剖面”表示的是某个时刻用户使用该应用的典型模式，一般由“用户执行的操作”、“执行不同操作的用户比例”以及“用户使用系统的频率”进行描述。测试场景模拟包括业务、业务比例、测试指标的目标以及需要在测试过程中进行监控的性能计数器。

测试场景可以是多个测试目标的综合体现，表 11-4 描述了一个测试场景的内容。

表 11-4　测试场景示例

场景名称	场景业务及用户比例分配	测试指标	性能计数器
用户登录	登录业务，100%用户 总用户数 200 人	响应时间 （<5s）	服务器 CPU Usage 服务器内容 Usage
标准日常工作	入账业务，40%用户 查询业务，30%用户 统计业务，30%用户 总用户数 200 人	响应时间 （入账<6s） （查询<5s） （统计<10s）	服务器 CPU Usage 服务器内容 Usage
⋮	⋮	⋮	⋮

场景可被看作用户实际运行环境的“剖面”，也就是说，场景体现的是用户实际运行环境中的具有代表性的业务使用情况。用户场景一般由用户在某一个时间段内所有业务的使用状况组成。

表 11-4 中描述的测试场景考虑了两种具有代表性的用户业务使用情况：一种是“用户登录”场景，该场景发生在用户每天上班之后的半小时内，此时的用户行为是“全部用户执行登录操作”；另一种是“标准日常工作”场景，该场景用于描述用户的日常工作，在这个场景中，用户执行三种不同的业务，“入账”业务的执行用户占 40%，“查询”和“统计”业务的执行用户占 30%。

（3）测试用例设计。

在设计完成测试场景之后，为了能够把场景通过测试工具体现出来，并能用测试工具顺利进行测试执行，因此有必要针对每个测试场景规划出相应的工具部署、应用部署、测试方法和步骤，这个过程就是测试用例设计活动。

测试用例是对测试场景的进一步细化，细化内容包括场景中涉及业务的操作序列描述、场景需要的环境部署等内容。以表 11-4 的“用户登录”场景为例，要将其细化为用例，就需要描述“登录业务”的具体步骤，例如：

“登录业务”步骤。

① 用户进入登录页面。

② 用户输入正确的用户名和口令。

③ 用户单击“登录”按钮。

④ 直到出现登录成功的页面，判断该页面成功显示的方法是 HTML 页面内容中的“欢迎”文本。

从描述中可以看到，在用例中，一个业务描述会被描述成操作的序列，并且在该序列中，一定会给出判断业务是否执行成功的准则。

一个较为复杂的性能测试用例设计内容如图 11-2 所示。

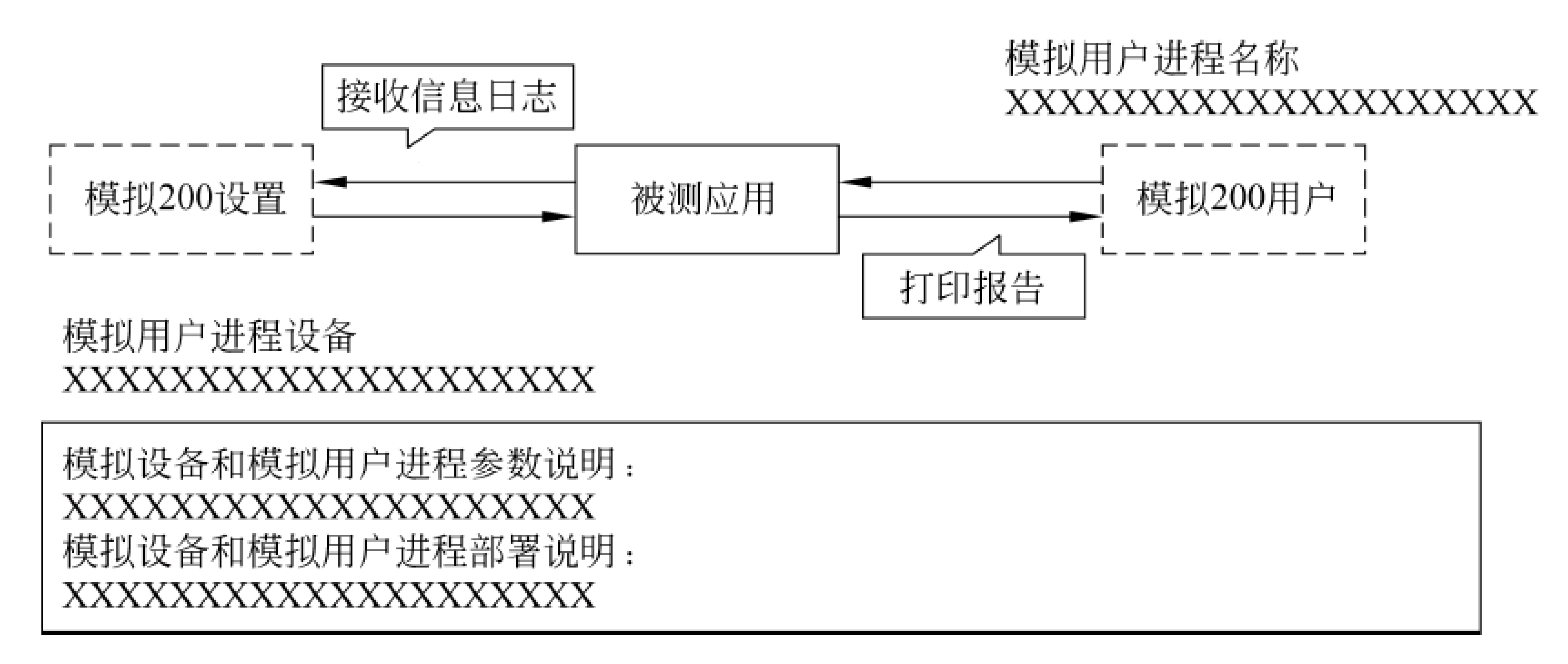


图 11-2　一个较为复杂的性能测试用例

该模型用于产生测试的负载，根据测试用例的描述，需要保持 200 个并发用户的操作维护负载，因此在本模型中，在“负载机 1”上启动了 200 个模拟设备的进程，在“负载机 2”上启动了 200 个模拟并发操作的客户线程(4 个进程)，每个线程以每分钟 1 条命令的频度发送命令。

测试方案如下：

① 时间同步方法。鉴于本测试的模拟设备、模拟 OMC 和 OMC 都运行在不同的机器上，本次测试的结果又是和时间精确相关的，因此需要一个精确的时间同步机制来保证所有参与测试的机器在时间上是一致的；在这一方案中，选择了一台 UNIX 主机作为时间同步的服务器，在其他 UNIX 主机上通过 ntp 进行时间同步；在 Windows 终端上通过开源的 network time 程序进行时间同步。

② 设备和用户模拟方法。用模拟程序 M1 模拟 200 个网元，M1 程序能接收用户下发的命令 CMD1、CMD2 并发送回应；用模拟程序 M2 模拟 200 个客户端进行连接，充当负载；M2 程序以每分钟一条命令的频率随机发送 CMD1 和 CMD2 命令。

③ 执行方法。实际运行一个被测应用，在被测应用中由用户手工输入命令，程序记录下用户输入命令时间等关键时间点。

④ 数据记录方法。为了记录时间 $T1$、$T2$、$T3$、$T4$，有以下约定。

a. M1 发送的命令附带发送时的时间戳。

b. 被测应用在发送命令时，附带一个用户输入命令结束的时间戳。这个时间就是所定义的时间 $T3$。

c. M1 记录接收到命令的时间 T1，并从接收到的命令中分离出时间 $T3$，记录 $T1$、$T3$ 和 $T3-T1$；M1 在发送回应的时候在回应的报文中附带发送时的时间戳（$T2$）。

d. M2 程序接收网元模拟程序发送的回应报文，分离并记录出其中的时间 $T2$、记录报文回显完成的时间 $T4$，并计算 $T4-T2$。

⑤ 测试持续时间。持续测试 1 小时，在 1 小时中通过被测应用发送命令。

(4) 脚本和辅助工具开发。

脚本和辅助工具的开发是测试执行之前的最后步骤，测试脚本是对业务操作的体现，一个脚本一般就是一个业务的过程描述。

除了脚本，测试辅助工具也需要在本活动中进行开发。在上面的例子中，M1 和 M2 都是用来辅助进行测试的辅助工具，通常这些工具的开发也放在该活动中完成。除了这类在测试中充当“桩模块”或是“驱动模块”的测试辅助工具，有时候还需要提供辅助进行服务器性能监控的脚本作为测试辅助工具。特别要说明的是，性能测试辅助工具的失效会影响测试结论，因此测试辅助工具需要在测试过程中进行妥善的管理，对其进行基本测试是必要的。

测试脚本的开发通常基于“录制”，依靠工具提供的录制功能，可以将需要性能测试关注的业务在工具的录制下操作一遍，然后基于录制后的脚本进行修改和调试，确保其在性能测试中能顺利使用。最常用的脚本修改和调试技巧是“参数化”、“关联”和“日志输出”。

11.2.5 测试执行与管理

测试和执行过程用于建立合适的测试环境，部署测试脚本和测试场景，执行测试并记录测试结果。

(1) 建立测试环境。

该活动用于搭建需要的测试环境。在设计完成用例之后就会开始该活动。该活动是一个持续性的活动，在测试过程中，可能会根据测试需求进行环境上的调整。

建立测试环境活动需要多个团队角色的参与，环境由测试设计人员设计完成，建立测试环境的活动由测试实施人员按照设计的要求组织建立，团队中的支持角色负责协助测试实施人员。

建立测试环境一般包括硬件、软件系统环境的搭建，数据库环境的建立，应用系统的部署，系统设置参数的调整，以及数据环境的准备等几个方面的工作内容。

一般来说，建立测试环境要遵循下列原则。

① 真实：尽量模拟用户的真实使用环境；

② 干净：测试环境中尽量不要安装与被测软件无关的软件；

③ 无毒：测试工作应该确保在无毒的环境中进行；

④ 独立：测试环境与开发环境相互独立。就是说开发环境和测试环境最好分开，即测试人员和开发人员分别用不同的服务器（数据库、后台服务器等），避免造成相互干扰。

测试环境的维护是一个比较困难的问题。性能测试中使用的数据量巨大，每次运行测试都可能产生大量的测试数据，而且性能测试可能需要部署大量的测试辅助工具和程序。为了保证测试结果的可比性，一般都需要在每次测试结束后恢复初始的测试环境，如果管理不善，这一恢复工作经常会引起很大的混乱。

在每次测试运行完成后，准备进行下一轮的测试运行之前，用检查列表（Checklist）来检查环境的可用性，参见表 11-5。

表 11-5　测试环境检查的检查列表

条目名称	检 查 内 容	责任人	维 护 方 法
硬件环境	硬件环境是否与拓扑描述一致		硬件拓扑结构图
软件环境	软件环境是否与软件环境列表中的描述一致 应用部署是否成功 测试辅助工具是否部署成功 软件参数设置是否符合要求		软件环境列表 应用部署检查 测试辅助工具部署检查 软件参数设置表
数据环境	数据是否与数据要求描述表中的一致 上次测试是否引入了额外的数据而没有清除		数据要求描述表 数据维护脚本或是录入（Import）方式

（2）部署测试脚本和测试场景。

在建立合适的测试环境之后，接下来的工作是部署测试脚本和测试场景。部署测试脚本和测试场景活动通过测试工具本身提供的功能来实现。

对脚本和场景的部署需要熟悉测试工具的人员来完成，在过程模型中，该活动由测试实施人员进行。在场景部署完成后，一般需要一个确认步骤，在该步骤中，测试设计人员确认场景部署与预期的设计一致。

部署活动最终需要保证场景与设计的一致性，保证需要监控的计数器都已经部署好了相应的监控手段。

（3）执行测试和记录结果。

准备好环境和部署好测试脚本以及场景后，就可以执行测试并记录测试结果了。在测试工具的协助下，测试执行是非常简单的操作，一般只需要使用菜单或是按钮就可以完成；记录测试结果也可以依靠测试工具完成，通过测试工具的监控（Monitor）模块，可以获取并记录需要关注的性能计数器的值。

在测试工具本身不提供对需要关注的性能计数器进行监控的功能时，可以用一些操作系统的工具，自行编制部分脚本解决这个问题。一般的方法是用脚本调用操作系统提供的工具，在脚本实现中将各性能计数器值分析出来并按照一定的格式记录在本地文件中。

11.2.6 测试分析

测试分析过程用于对测试结果进行分析，根据测试的目的和目标给出测试结论。

性能测试的挑战性在很大程度上体现在对测试结果的分析上。可以说，每次性能测试结果的分析都需要测试分析人员对软件性能、软件架构和各种性能指标具有相当程度的了解。

测试分析过程是一个灵活的过程，很难给出一种具体的、能适应各种性能测试需要的统一的过程活动列表。性能测试的分析需要借助各种图表，一般的性能测试工具都提供了报表模块来生成不同的图表，报表模块同时还允许用户通过叠加、关联等方式处理和生成新的图表。

性能分析的通用方法之一是“拐点分析”法。“拐点分析”法是一种利用性能计数器曲线图上的拐点进行性能分析的方法。该方法的基本思想基于这个事实：性能产生瓶颈是由于某个资源的使用达到了极限，此时的表现是随着压力增大系统性能表现急剧下降，因此，只要关注性能表现上的“拐点”，获得“拐点”附近的资源使用情况，就能够定位出系统的性能瓶颈。“拐点分析”法在确定引起系统瓶颈的系统资源方面能发挥一定作用，但由于它只能定位到资源上的制约，而不能直接定位到引起制约的原因，该方法还必须配合其他方法使用。

11.3 敏捷性能测试模型

敏捷开发方法是一组软件开发方法的集合，它鼓励协作、交互、面向可交付的产出，通过迭代的方式实现小步快走，目标是在需求不断变化的情况下能够按时交付满足用户需求的产品。敏捷开发在近年来被越来越多的公司和组织所接受，已经成为一种主流的开发方法。

“敏捷测试”这个术语通常指敏捷开发方法中测试相关的部分。相对于传统的软件开发过程，敏捷开发方法中的测试工作与开发工作联系更紧密，更注重建立对应用的多层次、多角度的测试标准。从具体的测试方法上来说，敏捷测试具有更大的灵活性，广泛采用探索式测试方法。探索式测试侧重对应用的探索，在执行测试过程中根据测试者的经验和应用的表现设计并执行新的用例。

性能测试具有探索式的特性，回顾一下执行性能测试的过程：设定性能测试场景和性能测试目标——使用性能测试工具对被测应用施加压力——观察应用系统在压力下的表现。在整个过程中，无法预期被测系统在给定负载下的具体性能表现，大多数情况下，都会根据系统在给定负载下的性能表现决定下一步的操作，一旦发现给定负载下某些可能的性能瓶颈，就会立即设计新的测试，以验证和确认性能瓶颈所在。因此，性能测试过程典型地具有探索式的特性。另一方面，随着应用系统的日趋复杂，仅在系统测试和验收测试阶段执行性能测试已经不能满足尽早发现和解决系统性能瓶颈的要求，性能需求可能要分解到应用单元（模块、接口，甚至是函数）中。在这种情况下，建立各层次的性能标

准并在各层次上开展性能测试就非常必要了。本节将从检查表、活动、环境与工具三个方面入手，讨论与敏捷方法结合的性能测试。

敏捷性能测试模型(APTM)不仅仅适用于指导敏捷过程中的性能测试开展，也同样适合在非敏捷的环境下尽早建立性能测试，尽早发现系统可能的性能问题。

11.3.1 APTM 的检查表

作为一种常用的工具，检查表可以为使用者提供框架和指导。敏捷注重过程的灵活性，因此，在采纳敏捷的 APTM 模型中，不倾向于定义严格的过程，而是使用检查表对敏捷性能测试进行指导。

APTM 的检查表可以看作敏捷性能测试的总体原则。APTM 检查表的内容体现了敏捷性能测试中的倾向：在迭代中设立性能目标，通过性能测试验证性能目标；在各个层面上建立性能测试；尽可能通过自动化的方式建立敏捷环境下的性能测试支持环境。

APTM 中常见的问题介绍如下：

(1) 在每个迭代中是否有明确的性能测试任务？

在每个迭代中设定明确的性能测试任务可以帮助开发团队明确每个迭代的性能标准，安排性能测试的时间。并非每个迭代中都需要在性能测试上花费许多时间，但不管怎样，在每个迭代中考虑性能测试任务(包括函数级别的性能测试)总是能够帮助开发团队更好地思考项目的性能要求。

(2) 每个迭代的验收测试标准中是否有性能验收测试标准？

敏捷的每个迭代都需要有明确的标准决定一个迭代是否可以结束，该标准就是迭代的验收测试标准。只有在验收测试标准中明确设置了具体的性能标准，性能要求才能在每个迭代中得到实现。可以在不同的迭代中设定不同的性能标准，“能够在 5s 内导入 1MB 的 XML 文件”是一个针对模块的合理的性能验收准则；“在 10 000 个用户的条件下能够在 3s 内完全展示应用首页”也是一个合理的性能验收准则。

(3) 是否在单元、接口和系统级别分别设置了相应的性能测试？

如软件性能工程(SPE)建议的那样，在不同层面建立性能测试可以帮助开发者尽早开展性能测试，并能帮助开发者将性能需求尽可能地分解到更小的层面，以帮助开发者更好地实现和管理应用的性能。APTM 建议至少在单元、接口和系统三个层面设置相应的性能测试。在单元层面，性能测试通过对函数的运行时间评估来进行，涉及的依赖通过 Mock 方式解决；在接口层面，性能测试要求设置接口运行环境，在少量用户的情况下检查接口的性能；在系统层面，则要求接近生产环境的性能测试环境，通过模拟真实的用户负载来进行测试。

(4) 是否已经建立合适的性能测试支持环境帮助实施各个层次的性能测试？

敏捷环境下的测试高度依赖自动化。同样，敏捷性能测试也需要良好的性能测试支持环境。图 11-3 给出了简单的性能测试支持环境的图示。一般而言，性能测试支持环境包括持续集成环境、性能测试运行环境、基准比较环境和测试环境管理几个主要部分。其中，性能测试运行环境由执行性能测试的工具构成，能够提供从单元级别的性能测试到系

统级别性能测试的测试执行支持;基准比较环境则与应用的分布方式相关,在应用每次发布时对其主要的性能指标进行验证,保证新发布版本的主要性能指标不比原有的版本差;测试环境管理则主要提供各个层面性能测试环境的建立功能,包括单元和接口级别的Mock工具、数据生成工具、测试环境备份和恢复工具等。

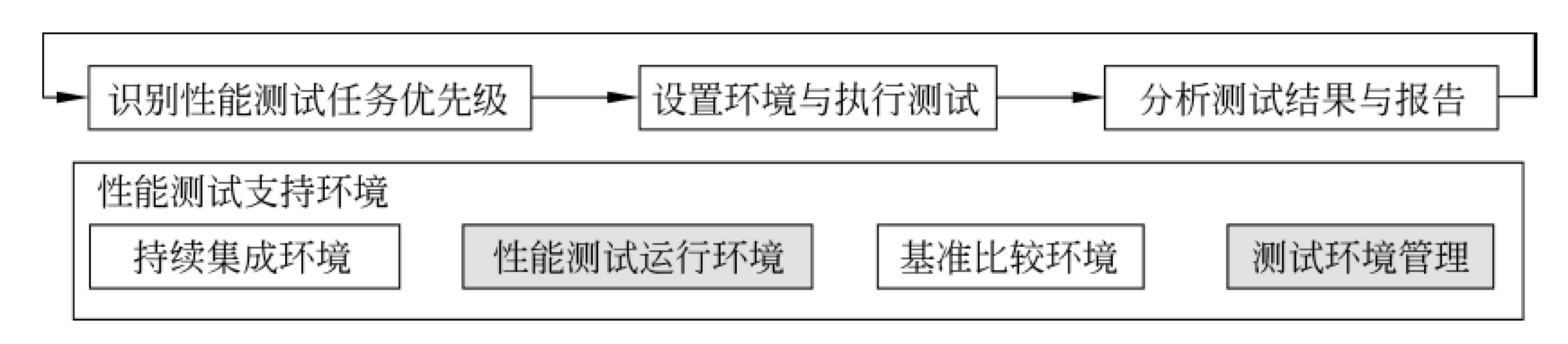


图 11-3 敏捷性能测试活动

(5) 性能测试结果报告是否包含在反馈体系中?

作为敏捷的核心价值观之一,反馈为敏捷开发团队提供了良好进度和质量评估机制,持续集成(CI)就是一个典型且有效的反馈机制。将性能测试结果报告包含到反馈体系中可以大大提高性能测试的价值,帮助开发团队更好地管理性能。将对主要性能指标的测试包含到持续集成环境中就是一个良好的实践。

11.3.2 APTM中的活动

敏捷性能测试使用迭代的方式管理性能测试,并倾向于在每一个层面上建立性能测试。如果考察每个迭代中的性能测试活动,则需要首先在每个迭代中识别性能测试任务的优先级,根据优先级顺序决定在当前迭代中包含哪些性能测试任务。在确定任务之后,需要设置测试环境并执行相应的性能测试。测试执行完成后,需要对结果进行分析并报告结果。一个迭代中的性能测试活动与传统的性能测试组织过程比较接近,不同的部分在于性能测试任务的识别。传统意义上的性能测试通常在系统或验收测试阶段开展,每次开展的性能测试必然有一个较为明确的目标(验证系统级别的性能是否符合性能需求,或是找到系统的性能瓶颈)。但在敏捷方法中,每个迭代中的性能测试任务都需要根据其价值进行选择,只有高价值的性能测试任务才会被放入迭代中。总体来说,APTM中的主要活动如下:

(1) 识别性能测试任务优先级。

(2) 设置环境与执行测试。

(3) 分析测试结果与报告。

(4) 在下一个迭代中重复步骤(1)～(3)。

另一方面,由于敏捷性能测试要求在每个迭代、每个层面上开展,因此必须具有良好的性能测试支持环境。

图11-3展示了由持续集成环境、性能测试运行环境、基准比较环境和测试环境管理几个部分组成的性能测试支持环境。

(1) 性能测试任务优先级。

识别任务优先级是敏捷开发的每个迭代中首先要做的事情。敏捷遵循“交付价值”原

则，在每个迭代中需要根据其产生的价值决定加入哪些具体的性能测试任务。敏捷开发方法中用于估算和确定优先级的方法都可以用来帮助识别和确定性能测试任务的优先级。

如前所述，是否在迭代中包含某个性能测试任务是由该任务的价值决定的。为了准确评估任务的价值，首先需要了解项目的上下文、被测系统以及性能测试的目标，可通过以下问题更好地了解。

① 客户的期望是什么？客户希望以怎样的方式验证性能？客户是否希望在每个迭代中对性能进行评估？

② 发布流程是怎样的？性能测试需要关心哪些构建（持续集成、日构建、转为性能测试发布的构建、发布构建）？

③ 性能目标是如何被分解到每个迭代中的？开发工程师会在单元测试中包含性能验证吗？

④ 在本迭代中，团队最关心的事情是验证性能、量度性能，还是对性能进行调优？

⑤ 迭代中的任务优先级是如何评估的？

当然，之前章节中对性能测试应用领域的分析同样适用于在 APTM 中识别性能测试的目标。

（2）设置环境与执行测试。

11.2 节中讨论的性能测试环境设置与测试执行的相关内容同样适用于 APTM 模型。由于 APTM 模型中涉及的性能测试包括单元、接口和系统多个层面的性能测试，而单元与接口层面的性能测试使用的工具和技术与系统级别的性能测试有所不同，之后的章节将详细讨论 APTM 中使用的工具。

（3）分析测试结果与报告。

对测试结果的分析依赖与具体的测试目标。对于细粒度的性能测试，如函数级别的性能测试，由于其环境简单，测试目标单一，其结果分析非常简单。但对系统级的性能测试，测试分析过程是一个灵活的过程，很难给出一种具体的、能适应各种性能测试需要的统一的过程活动列表。

在 APTM 中，性能测试结果报告通常包含在项目的反馈体系中，通过在持续集成、验收测试等反馈环节包含性能测试报告，开发组可以准确地了解每个迭代，每次提交是否带来了对性能需求更好的实现。

11.3.3 环境与工具

敏捷性能测试中的活动通常需要工具与环境的支持，下面对具体的环境和相应工具进行说明。

（1）持续集成环境。

持续集成环境并非是为敏捷性能测试而专门设置的。在敏捷体系中，持续集成是一种最佳实践，提供了对产品质量的持续评估与反馈体系。Martin Fowler 将持续集成定义为“一种软件开发实践，即团队开发成员经常集成他们的工作，通常每个成员每天至少集

成一次，也就意味着每天可能会发生多次集成。每次集成都通过自动化的构建(包括编译、发布、自动化测试)来验证，从而尽快发现集成错误。许多团队发现这个过程可以大大减少集成问题，让团队能够更快地开发内聚的软件。”简而言之，持续集成通过持续构建的方式保证开发工程师的代码能够经常性地进行集成，在集成过程中尽可能早地发现问题。对敏捷性能测试来说，持续集成通过及时的反馈，保证性能测试成为持续的产品质量评估体系的一部分，从而使性能测试在敏捷过程中发挥更大的价值。

(2) 测试执行环境。

性能测试执行需要特定工具的支持。敏捷性能测试由于覆盖了多个层面的性能测试，需要的性能测试工具也更多样。这些工具都能在敏捷性能测试中针对某个层面的性能测试发挥作用。此外，在单元级别和接口级别，能够发挥作用的工具还有很多。

① 单元层面的性能测试工具。

单元层面的性能测试主要用来对应用的函数或模块进行性能评估。对算法的性能评估就是该层面性能测试的典型例子。根据使用的编程语言不同，单元层面的性能测试工具有不同的选择。以 Java 语言为例，从 JUnit4 开始，JUnit 工具就提供了函数级别的性能测试支持。在 JUnit4 中，通过 Timeout 标签可以达成对性能测试的支持。例如下面的这段示例代码，如果函数执行的时间超过 500ms，该 JUnit4 测试就会失败。

```
@Test (timeout=500) public void retrieveAllElementsInDocument(){
    Doc.query("//*")
}
```

除了 JUnit，还有其他一些类似的工具为其他编程语言提供类似的支持。需要说明的是，单元层面的性能测试往往需要 Mock 工具的支持。Mock 工具可以减少应用对环境的依赖性，并增加性能测试的可靠性。

② 接口层面的性能测试工具。

用于接口级别性能测试的工具也有众多选择，依赖于应用接口的不同类型，需要选择不同的工具。例如，HTTP 接口的性能测试工具包括 curl-loader 等；SOAP 接口的性能测试工具包括 SoapUI 等；REST 接口的性能测试工具可以使用 RestClient 库和 JMeter 工具建立。

③ 系统层面的性能测试工具。

系统层面的性能测试工具的选择面相对较窄，商业工具 LoadRunner、开源工具 JMeter 以及前端的性能测试工具就属于此类。

(3) 基准比较环境。

所谓基准比较，是指基于构建建立的一套比较机制。例如，可以为应用的发布版本建立一个基准性能测试，在相同的测试环境和负载下评估每个发布版本的主要性能指标，各个版本的主要性能指标构成的曲线可以清晰地显示应用的性能变化趋势(变好还是变坏)，甚至可以作为版本是否能够发布的评价标准。

为什么需要基准比较？考虑敏捷性能测试中各个层面的性能测试，在系统层面，为系统建立性能的接受准则并不困难(用户故事)，但在接口和单元层面，也许并不能在每个迭

代中准确定义其可接受的标准。在这种情况下,保证“下一个版本的性能至少不比上一个版本更差”是一个可行的策略。该策略能够保证应用的性能一直具有变好的趋势。

图 11-4 展示了某接口的基准测试结果。

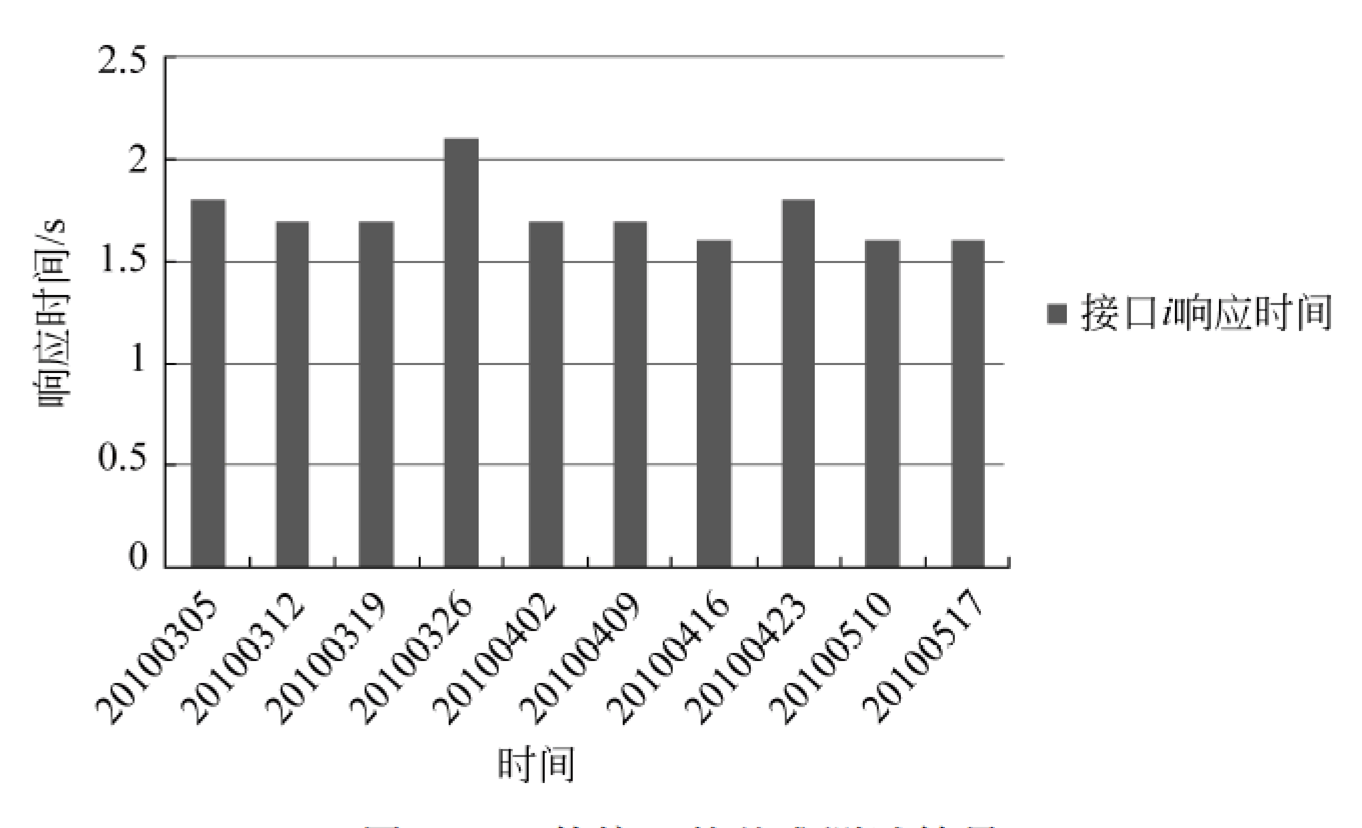


图 11-4　某接口的基准测试结果

(4) 测试环境管理。

敏捷性能测试中的测试环境管理同样需要对性能测试所需的软件和硬件环境进行维护,需要与产品的发布过程集成,需要使用自动化方式快速建立和恢复测试环境。在测试环境管理的具体方法上,敏捷性能测试与传统性能测试采用的方法相同。

11.4　本章小结

本章介绍了性能测试团队中人员的构成,以及两种性能测试模型:PTGM 与 APTM。PTGM 是典型的可用于系统级别的性能测试,遵循该过程模型,可以在系统级别上完成性能测试;而 APTM 模型是结合了敏捷思想的性能测试模型,与 PTGM 不同,该模型并不强调过程,并且从检查表、活动和工具三个方面介绍了 APTM 模型的特点和应用。

第 12 章　性能测试工具原理

广义地讲，可以把性能测试过程中使用到的所有工具都称为性能测试工具，性能测试工具分为两大类，服务端性能测试工具和前端性能测试工具；服务端性能测试工具需要支持产生压力和负载，录制和生成测试脚本，设置和部署场景，产生并发用户和向系统施加持续的压力；而前端性能测试工具则不需要关心系统的压力和负载，只需要关心浏览器等客户端工具（目前的前端性能测试工具主要是 Web 前端性能测试工具）。

12.1　服务器端性能测试工具架构

图 12-1 给出了服务器端性能测试工具的架构。一般来说，服务器端性能测试工具包括以下部件。

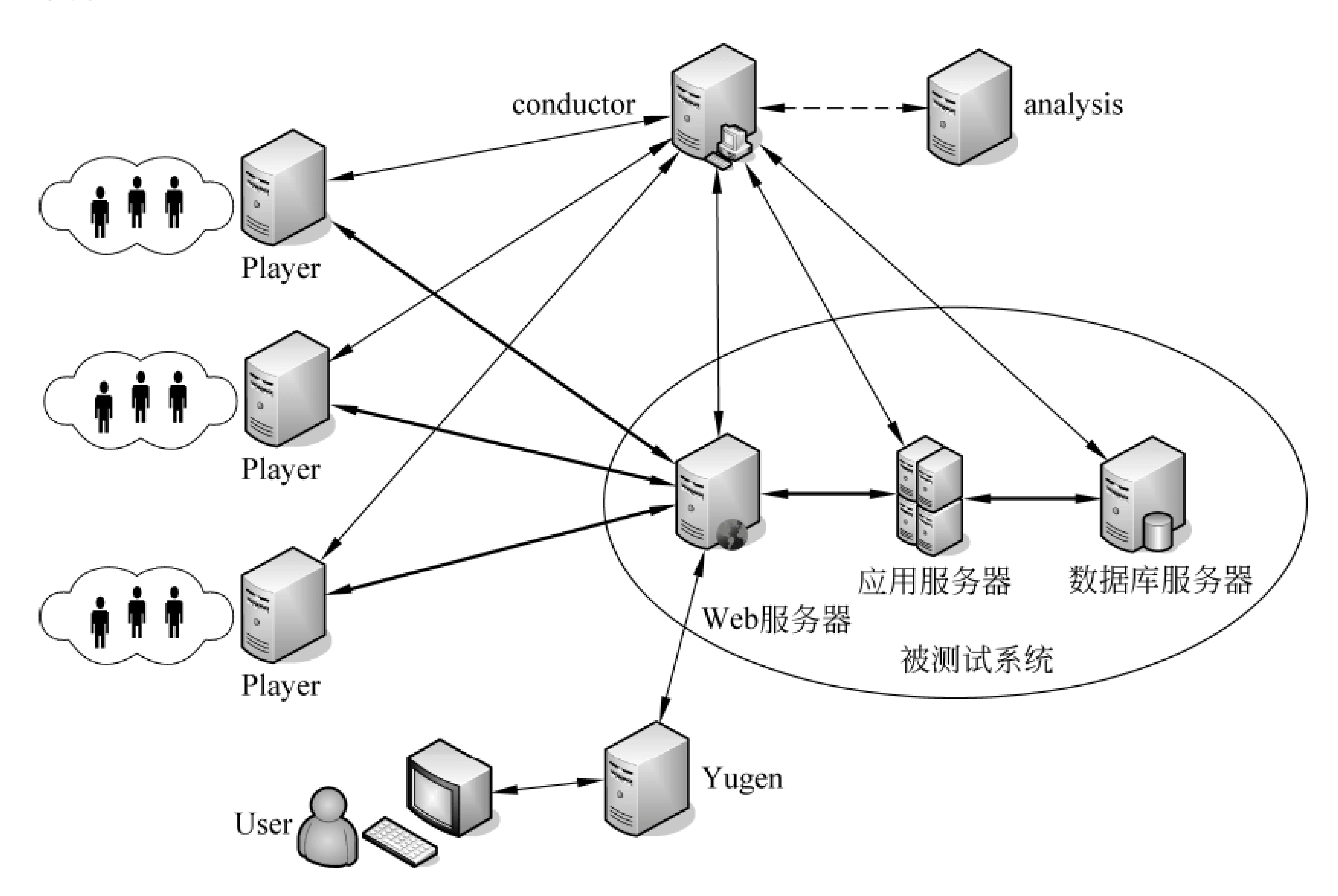


图 12-1　服务器端性能测试工具架构

（1）虚拟用户脚本生成器（Virtual User Generator）；
（2）压力产生器（Player）；
（3）用户代理（Agent）；
（4）压力调度和监控系统（Conductor）；
（5）压力结果分析工具（Analysis）。

1. 虚拟用户脚本生成器

虚拟用户脚本生成器的常用模式是通过 Proxy 方式实现的，但是目前已经有其他的模式，例如 LoaderRunner 中就以端口监听的方式实现。

对于 Proxy 方式，具体来说，就是通过一个 Proxy，该 Proxy 作为客户端和服务器端之间的中间人(Middle Man)，接收从客户端发送的数据包，记录并将其转发给服务器端；接收从服务器端返回的数据流，记录并返回给客户端。这样，无论是客户端还是服务器端都以为自己在一个真实的运行环境中，而虚拟脚本生成器能通过这种方式截获并记录客户端和服务器端之间的数据流。

对于端口监听方式，测试工具会自动监控用户所指定的 URL 或应用程序所发出的请求及服务器返回的响应，它作为一个第三者监视着客户端与服务器端的所有对话，然后把这些对话记录下来，生成脚本，再次运行时模拟客户端发出的请求，捕获服务器端的响应。它在做监视时会自动捕捉客户端发出请示时所用的端口，并根据请求内容向服务器端的相应端口发送，而服务器回应时，根据请求消息中的端口向客户端某个端口发出回应，也就是说监听的端口是由应用程序或请求所决定的。

当然，截获数据流仅仅是虚拟脚本生成器的第一个步骤，在截获数据流之后，虚拟脚本生成器还需要根据录制时选择的协议类型，对数据流进行分析，然后，用脚本函数将客户端和服务器端之间的数据流交互过程体现为脚本的语句。下面的例子是用性能测试工具 LoadRunner 录制生成的脚本片段(HTTP 协议)：

```
…
    web_add_cookie("PREF= ID= 235bcabf6ffeabaf:NW= 1:TM= 112082384
6:LM= 1124062513:GM= 1:S= kvXkwVuAjnjlDlRN;
DOMAIN= www.google.com");

    web_add_cookie("rememberme= false; DOMAIN= www.google.com");

    web_url("www.google.com",
        "URL= http://www.google.com/",
        "Resource= 0",
        "RecContentType= text/html",
        "Referer= ",
        "Snapshot= t1.inf",
        "Mode= HTML",
        LAST);

    lr_think_time(11);

    web_submit_form("search",
        "Snapshot= t2.inf",
        ITEMDATA,
```

```
        "Name=q", "Value=软件测试工具", ENDITEM,
        "Name=lr", "Value=", ENDITEM,
        "Name=btnG", "Value=Google 搜索", ENDITEM,
        LAST);
...
```

在这个例子中，访问了 http://www.google.com 网站，并在搜索引擎的输入框中输入“软件测试工具”作为搜索关键字。从这段示例代码中可以看到，虚拟用户脚本生成器在截获数据流后对其进行了协议层上的处理，最终形成的是人们能很容易看懂的 HTTP 业务交互过程脚本。

除了能够录制应用之间的通信数据流生成脚本之外，虚拟用户脚本生成器一般还自带 IDE 环境，用户可以通过该 IDE 环境对脚本进行修改和调试。对脚本的修改和调试中，最常见的三种技巧是参数化、关联和 Log 输出。

2. 压力产生器

压力产生器用于根据脚本内容，产生实际的负载。在性能测试工具中，压力产生器扮演着“产生负载”的角色。例如，一个测试场景要求产生 100 个虚拟用户（VU），则压力产生器会在调度下生成 100 个进程或者线程，每个线程都会对指定的脚本进行解释执行。

设想这样一种情况：性能测试要求 1000 个 VU 共同进行。无论采用进程还是线程作为压力产生的方式，它们都需要一定的系统资源，一般来说，一台具有 512MB 内存的 PC 可以顺利运行 200 个左右的 VU，但对需要 1000 个 VU 的情况，显然很难指望通过一台 PC 产生如此多的 VU。这时，唯一的解决方案就是通过多台机器进行协作，但多台机器之间如何产生“步调一致”的 VU 呢？答案就是“用户代理”。

3. 用户代理

用户代理是运行在负载机（Load Machine）上的进程，该进程与产生负载压力的进程或是线程协作，接收调度系统的命令，调度产生负载压力的进程活线程，从这个意义上说，用户代理也可以被看作压力产生器的组成部分。

用户代理一般以后台方式在负载机上运行。

4. 压力调度和监控系统

压力调度和监控系统是性能测试工具中直接与用户交互的主要内容。压力调度工具可以根据用户的场景要求，设置各种不同脚本的 VU 数量，设置同步点等，而监控系统则可以对各种数据库、应用服务器、服务器的主要性能技术器进行监控。

例如，给出一个典型的用户场景的描述：80 个用户以脚本 1 的方式对系统进行访问操作，120 个用户以脚本 2 的方式对系统进行访问操作，用户数按照每分钟增加 10 个的方式增长，对这个场景，可以通过压力调度系统调度 80 个 VU 以脚本 1 的方式产生负载，同时 120 个用户以脚本 2 的方式产生负载，把 VU 的增长模式设置为“每 60s 增加 10 个 VU”。

不同的性能测试工具可以提供对不同类型的服务器性能计数器监控的能力。是否具有强大的性能计数器监控能力通常也是衡量性能测试工具的功能是否完备的指标之一。以 LoadRunner 为例，其可以对大部分数据库服务器、应用服务器和服务器主机的性能指标进行监控。

5. 压力结果分析工具

压力结果分析工具可以用来辅助进行测试结果的分析。性能测试工具附带的分析工具一般都能将监控系统获取的性能计数器值生成曲线图、折线图等图标，还能根据用户的需求建立不同曲线之间的叠加、关联操作，从而提供从各方面揭示压力测试结果的能力。

仍然要强调的是，压力结果分析工具本身不能代替分析者进行性能结果的分析，最多只是提供多种不同的数据揭示和呈现方法而已。对这些数据进行分析必然要依靠测试工程师对系统性能分析的经验和知识。

12.2 性能测试脚本录制时的协议类型

本章的开头提到，确定性能测试脚本录制时使用的协议类型经常是一个容易引起误会的问题。根据实际情况，很多测试工程师在刚接触性能测试时，常常会想当然地根据开发语言等来决定协议的选取，导致录制后的脚本不能回访成功。

表 12-1 是本书总结的一个表格，该表格以 LoadRunner 工具下的情况为例大致给出了不同类型的应用在选择性能测试脚本录制时的协议的考虑方法。

表 12-1 性能测试脚本录制时的协议类型

应用类型	应 用 特 点	建议选用协议	备 注
Web 应用	应用采用 ASP 结构、J2EE 或是 dotNet 架构	HTTP/HTTPS 协议	Web 应用一般采用 HTTP/HTTPS 协议进行性能测试脚本录制，但特别要指出的是，有些借助客户端运行的组件扩展功能的 Web 应用，其客户端组件采用自定义 Socket 或是其他协议与服务器进行通信，此时需要在录制时选择多种协议
C/S 应用	客户端程序以 ADO、OLEDB 方式连接后台数据库	根据后台数据库类型选择相应的协议	例如，如果后台数据库是 Oracle，则在录制时选择 Oracle 协议
	客户端程序以 ODBC 方式连接后台数据库	ODBC 协议	
	客户端和服务端之间通过自定义的 Socket 协议进行通信	Socket 协议	

续表

应用类型	应 用 特 点	建议选用协议	备 注
C/S 应用	其他协议	根据具体协议类型进行分析	例如，有些应用为了能够适应复杂的广域网环境，采用 HTTP 协议作为 C/S 结构应用的客户端和服务端之间的通信协议，此时可以根据具体的协议来选择录制时使用的协议
组件	COM/DCOM	COM/DCOM 协议	这里提到的是针对组件的测试，商业性能测试工具一般提供了一种直接测试组件接口性能的方法
	EJB	EJB 协议	
服务	Web Service	Web Service 协议	有些读者可能会用 HTTP 来录制对 Web Service 服务进行性能测试的脚本，建议使用专门的 Web Service 协议进行录制
	Mail 服务器	SMTP 和 POP	
	FTP 服务器	FTP	
	其他	根据具体的协议选择最接近的录制协议	
应用服务器	Oracle Application Server	Oracle Application Server 协议	
	SAP	SAP	
	Tuxedo	Tuxedo 协议	
	其他	根据具体的协议选择最接近的录制协议	

在选择录制性能测试脚本的协议时，有几点必须说明的内容：

(1) 使用 Socket 协议可以对任何类型的应用通信进行录制，但这种录制生成的脚本很可能没有任何意义。举个简单的例子：假设有一个 C/S 结构的数据库应用，选择 Socket 协议固然可以生成脚本，但脚本中的内容不具有数据库流，而这种数据库流往往会随着环境的细微改变而进行相应的调整，这样，录制后的脚本就只能在非常特定的环境(软硬件环境、时间环境)下发挥作用，失去了脚本本身必须具有的适应性。

(2) 在对通信进行录制生成脚本后，对脚本进行回放，有时会出现回放无法继续的情况(停留在某个步骤无法进行下去)，此时应该考虑是否使用了合适的协议。很可能是由于协议选择不正确或是不全面，导致部分通信没有录制成功。

12.3 性能测试工具的选择与评估

对需要进行性能测试的组织来说，决定使用哪种性能测试工具也不是一个简单的问题。这个问题通常会有两个层面的意义：第一，创建还是购买？第二，如果购买，如何选择一种商业的工具？

12.3.1 创建还是购买

“创建还是购买”是一个不好回答的问题。实际上，商业化的工具有自身的优点，例如，其稳定性好，适用性比较广，而且总体的拥有成本比较低。但它也有自身的缺点：学习培训成本较高，某些特殊的需求不能满足等。确定创建还是购买，最好能够综合多个方面的需要进行决策。如果需要的是一个仅用于本次项目测试的工具，或是该被测系统使用了比较特殊的协议等原因，可以考虑自行创建所需的测试工具；否则，可以考虑选择购买已有的商业测试工具。表12-2给出了创建和购买各自的优势和劣势。

表12-2 创建测试工具和购买测试工具的比较

创　建	购　买
能够开发出最适合应用的测试工具	依赖于工具本身提供的特性，较难扩展
能够部分弥补被测试系统的不可测试性	只能从被测应用角度增强设计
易于学习和使用	依赖于工具的易用性和所提供的文档
工具的稳定性和可靠性不足	稳定性和可靠性有一定的保证
总体拥有成本高	总体拥有成本低
可形成组织特有的测试工具体系	很难与其他产品集成

总之，“购买”方式可以以较低的总体成本快速获得可用的软件，但如果被测对象本身有一定的特殊需求，最好是使用“创建”方式构造适合的测试工具。

“创建”方式的最大问题是成本和工具的稳定性。与已有的商业测试工具比较，自行创建的测试工具一般存在稳定性和可靠性方面的问题，甚至在使用工具测试发现问题后，还需要确定该问题是由工具引起的，还是由系统故障引起的。另外，自行创建的工具由于其适应范围较小，通常只能用于少部分的特定项目，因此，从总体拥有成本上来说，自行创建的工具通常具有较高的成本。

12.3.2 测试工具的评估和选择过程

图12-2描述了测试工具的评估和选择过程。

测试工具的评估可被概要描述为“从很多可用的工具中选择一个工具”，对于商业化的性能测试工具，一般可以按照下列的过程进行评估：

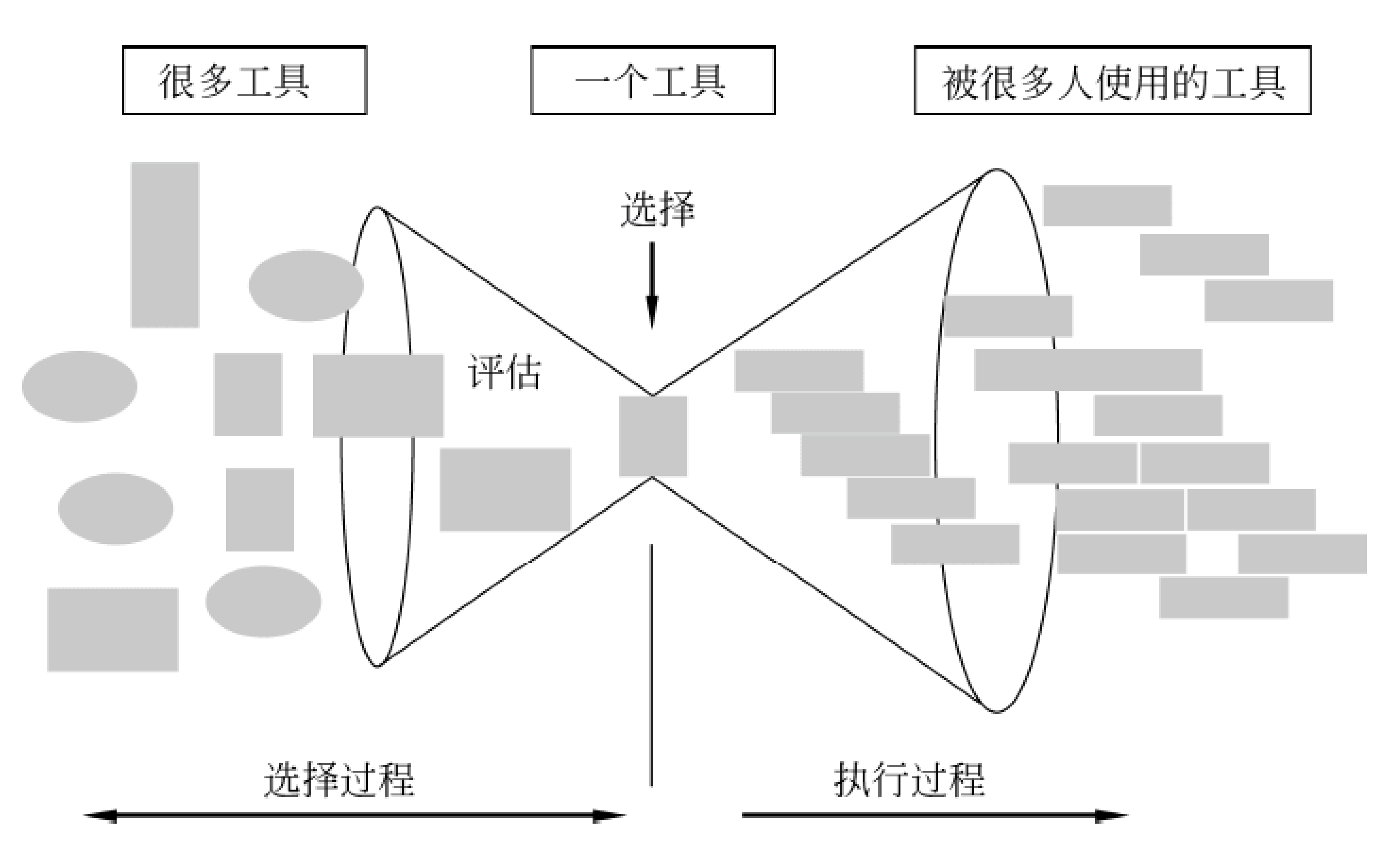


图 12-2 测试工具的评估和选择过程

(1) 列出需要的工具功能列表。

这个步骤以列表的方式给出所需要的测试工具的功能列表，该步骤有利于在不同的测试工具中发现“最适合自己的”，而不是“具有最多功能的”。软件测试工具厂商在推销自己的产品时，一般会强调自己的产品是“功能强大的”，但强大的功能同时也意味着更高的价格。如果从一开始就明确自己的需要，就可以避免为不必要的功能买单，或是在购买之后才发现工具不能支持测试需要。

可从以下几个方面考虑需要的功能：

① 工具能支持被测系统运行的平台(软硬件环境、数据库环境)吗?

② 工具能支持被测试系统使用的协议吗?

③ 工具是否能够支持特殊要求，例如防火墙、负载均衡、动态页面的生成等?

④ 工具能够提供对人们关心的服务器、应用服务器或是数据库类型计数器的监控吗?

⑤ 工具使用的脚本语言功能完善吗?

(2) 工具比较。

该步骤的主要目的是比较所有可供选择的工具，比较内容包括以下几个方面：

① 功能比较。评价各种不同工具与所需要的工具的功能的符合程度，可以在列出需要的工具的功能时，一并给出每个需要功能的优先级权重，这样在进行符合度评价时，可以得出每个工具的功能符合度加权值，利用该加权值进行比较。

② 工具能获取支持的比较。这个方面的比较包括：开发商或是代理商对工具的支持力度如何？该产品的平均更新周期有多长？是否能方便地在应用中获取工具的使用支持？产品的支持方式包括哪些？时效性如何？

③ 供应商的信誉。这方面的信息主要通过其他客户对供应商的评价来获得。

表 12-3 给出了一个可用于从功能方面评估性能测试工具的表格。

表 12-3 创建测试工具和购买测试工具的比较

功　能	功 能 子 项	得分(每项满分为 5 分)		
		适用性	符合性	可靠性
脚本	脚本录制			
	脚本编辑			
	脚本调试			
	可供选择的脚本语言			
	事务功能			
	集合点功能			
	参数化功能			
	模拟 Think Time			
	关联功能			
支持协议	HTTP/HTTPS 协议			
	……①			
支持监控	响应时间、吞吐量等			
	应用服务器监控			
	数据库监控			
	服务器监控			
	……②			
场景	场景类型			
	场景设置			
	场景执行			
	场景监控			
数据分析	图表类型			
	图表组合和关联			
	原始数据视图			
	页面时间分解			
其他功能	证书支持			
	外部扩展能力			
	网络带宽模拟			

注：① 可根据实际需要的协议进行评估。

② 可根据实际需要监控的类型进行评估。

(3) 成本分析。

工具的价格是成本的主要构成,工具的价格和产品的 License 方式直接相关。评估成本的时候,最先要弄清楚的就是工具的 License 方式。

不同的性能测试工具会采用不同的 License 方式。可别小看了这些 License 方式之间的差异,这种差异很可能会直接影响到拿到工具的价格。以目前最常用的 MI 的性能测试工具 LoadRunner 和 Seague 的 Silk Performer 为例说明这些差别。

LoadRunner 的 License 方式比较灵活,License 按照协议、VU 数量(可视为允许的最大并发用户数量)、性能监控器对象分别授予,例如,要针对一个 Web 应用进行测试,需要使用的协议可能是 HTTP/HTTPS 协议,测试要求最大的并发用户数为 1000,应用服务器使用的是 WebLogic、数据库使用的是 Sybase,则必须要在这个基础上才能按照 MI 的 License 方法得出一个最终的价格。

Silk Performer 的 License 方式则相对简单一些,该产品分为 Lite 和 Full 两个版本,Lite 版本只支持 Web 应用的测试,Full 版本支持更多的应用协议,再就是按照 VU 数量进行不同的 License 授予。另外,为了支持短时间内的项目性能测试,Silk Performer 还提供了一种"临时性"的 License,只允许在指定时间内使用 Silk Performer 工具。

在了解工具 License 方式的基础上,就可以根据需要选择产品部件,估算需要支付的产品价格了。

工具的学习曲线和必要的培训成本也是必须要考虑的一个因素。永远不要相信测试工具供应商做出的"我们的工具非常简单易学,只需要很短的时间就能让你学会"这种承诺。客观来说,目前的商业工具在产品架构和界面可操作性方面差别不大,学习曲线主要从系统支持的脚本语言类型是否被使用者熟悉,系统测试手册以及产品开发商或是代理商是否提供培训支持等入手。

12.4 本章小结

性能测试工具是性能测试过程中必不可少的帮手,但是目前有些不好的现象是:部分测试工程师,以为测试工具就是全部,以为熟悉了对测试工具的一些操作即可,这是远远不够的。作为一名合格的性能测试工程师,不仅要学会对测试工具的操作,更要理解测试工具的原理、作用和局限性。

在性能测试的过程中,性能测试工具的应用是否合乎实际需求,常常决定着测试的成败。因此对性能测试工具的了解程度就显得尤为重要。

本章以著名的性能测试工具 LoadRunner 为例,描述了性能测试工具的一般架构、工具的组成部分以及各部分的作用和工作原理,并且给出了如何选择性能测试工具的建议。

第 13 章　性能测试需求分析

需求分析是个繁杂过程，而性能测试需求除了要对系统的业务非常了解，还需要有深厚的性能测试知识，才能够挖掘分析出真正的性能需求。

性能需求分析是整个性能测试工作开展的基础。在这一阶段，性能测试人员需要与需求人员(客户)、领导及项目相关人员进行沟通，同时收集各种项目资料，对系统进行分析，确认测试意图。

测试需求分析阶段的主要任务是确定测试策略和测试范围。策略主要根据软件类型以及用户对系统性能的需求来定，测试范围则主要对系统的功能模块进行调研与分析，最终确认明确的需求。

13.1　制定负载测试的目标

进行测试需求分析，首先要明确性能测试目的，测试目的的不同将直接影响测试策略的制定。通过与用户的沟通，并结合对被测软件的了解，归纳出软件的结构特征、用户应用特征、用户关注内容等。

负载目标关系整个测试的场景设计、并发配比、结果评判，因此确定负载目标也决定了测试的总体方向。负载目标应该是系统被测对象的整体目标。

通常，客户会提出一些对系统的需求和期望，如搜索事务应该足够快并不允许失败，更新事务在工作的高峰时间能正常工作，在第二年期待系统能够承受两倍的用户量，系统在 1000 个并发用户下不会中断等。通过了解业务需求，负载目标都会转化为一系列具体的数值。典型的负载目标有响应时间、系统吞吐量、系统资源利用率等，一般可从两方面来划分。

前端：业务人员更关注前端并发用户数量或在线用户数量，以人数衡量。

后端：技术人员更关注后端应用服务器和数据库服务器的负载能力，以 TPS 衡量。

前端并发用户数量的计算在业界中有很多公式和原则，如 2/8 原则、10%在线用户数量估算、(在线用户数量×session 时间)/监控时间等，但各公式和原则计算出的并发用户数量并不精确，如有 10 万在线用户的系统不能说仅测试 10 万×10%＝1 万并发用户即可。

后端 TPS 反映被测应用的实际负载能力，对已有具体业务量的应用可以计算精确，如银行系统中某省行对公交易量日均 10 万笔，则可精确计算出 TPS 均值＝10 万/(6×3600)＝4.63 笔/秒(对公业务按 6 小时计算)，若被测应用达不到 TPS 要求则完成不了当日业务。

同一个被测应用以不同的视角估算负载目标，得到的数值可能会有很大的差异，因此如何正确选择负载目标，将会直接影响之后的测试方法和场景设计。

单从最终测试指标来说，对于一个软硬件环境固定的应用程序，只有一个负载指标是固定的，那就是最大事务处理能力，通常以 TPS 衡量。因此性能测试的目标就是确定被测应用的最大事务处理能力。

常规情况下，测试目的可以总结为以下三类：

(1) 主要验证软件性能是否符合软件开发合同或软件需求文档中的要求，是否符合预定的设计目标，是否符合用户现在或将来的应用要求等。总之，是验证软件性能与某项已知(或潜在的)标准或要求明确的性能指标的符合程度，可称之为性能符合性验证。

(2) 主要是在无明确性能指标的情况下，以了解软件的整体性能状况为目的，如软件所能支持的最大并发用户数，软件在不同环境下的性能极限。软件性能随用户数量的变化情况，软件性能随环境变化的情况，软件所能支持的最大数据量、支持的最低运行环境等，可称之为软件性能能力验证。

(3) 主要是为了通过性能测试找出软件系统的性能瓶颈，分析软件性能缺陷的原因，并进行针对性的性能优化，以改进软件性能，称之为性能调优。性能调优是性能能力验证测试工作的进一步深化，不仅包含了性能测试部分，还包括了性能调整部分。

13.2 收集系统信息

收集系统有用信息可以协助人们决定哪些业务流程需要录制，分析系统的最大负荷以及产生最大负荷的时间段，并确定有效的测试数据以及用户动作。系统有用信息的来源大致有以下几种。

(1) 有经验的用户：通过与有经验的用户交流，可以大致了解用户能够接受的响应时间和用户习惯。

(2) 系统管理员：对于已有具体业务量的系统，可以了解系统运行时的资源利用率等情况。

(3) IT 文档：这是性能测试需求的主要来源，项目开发计划书、需求规格说明书、设计说明书、测试计划等文档都可能涉及性能测试的要求，通过收集这些资料，可以找到初步的性能需求。相关的项目干系人有客户代表、项目经理、需求分析员、系统架构设计师、产品经理等。

(4) 在线统计：通过参考在线统计的历史数据，可以了解系统的使用情况，如每月、每星期、每天的峰值业务量是多少，用户以什么样的速度在递增中，用户对系统的哪些功能模块使用得最多，他们所占的比例等信息。

13.3 制订测试计划

确定明确的需求之后，要做的工作就是制订性能测试计划。

测试计划的大体内容如下。

(1) 项目的简单背景描述：本次性能测试的需求与目的，性能需求分析的结果是什么。

(2) 测试环境的准备：需要什么样的软硬件配置,网络状况登录。

(3) 测试数据的准备：对于某些性能测试是需要事先准备测试数据的。

(4) 测试的策略：前面进行需求分析的目的是制定测试策略,也就是设计符合需求的测试场景,需要对系统的哪些业务模块进行测试,如何进行？需要设计哪些场景以及设计这些场景的目的。

(5) 人员配备,如需要开发、DBA、运维等人员的参与协助。

(6) 性能测试的时间安排。

13.3.1 性能测试需求

性能测试需求是应用需求的衍生,而且测试用例也必须覆盖所有的测试需求,否则,这个测试过程就是不完整的,主要有以下几个关键点：

(1) 测试的对象是什么,例如：被测系统中有性能需求的功能点包括哪些;测试中需要模拟哪些部门的用户产生的负载压力等问题。

(2) 系统配置如何,例如：预计有多少用户并发访问;用户客户端的配置如何;使用什么样的数据库;服务器怎样和客户端通信;网络设备的吞吐能力如何,每个环节承受多少并发用户等问题。

(3) 应用系统的使用模式是什么,例如：使用在什么时间达到高峰期;用户使用该系统是采用 B/S 运行模式吗等问题。

针对用户提出的问题,做一个简单的需求问答,如表 13-1 所示。

表 13-1 用户需要与测试目标

测 试 目 标	用 户 需 求
测量对最终用户的响应时间	要花多少时间做完一笔交易
确定最优硬件配置	什么样的配置提供了最好的性能
检查可靠性	系统在无错情况下能承担多大及多长时间的负载
检查软硬件升级	这些升级对系统性能的影响有多大
评估新产品	服务器应该选择哪些硬件与软件
测试系统负载	在没有较大性能衰减的前提下,系统能够承受多大负载
分析系统瓶颈	哪些因素降低了交易响应时间

需求分析的方法有以下几种：

(1) 任务分布图方法。

使用任务分布图方法应关注以下两点：

① 有哪些交易任务。

② 在一天的某些特定时刻系统都有哪些主要操作。

举例如图 13-1 所示,可以得到：

① 12:00,交易混合程度最高。

② 10:00—12:00 是登录高峰期。

③ 数据更新业务的并发用户数最大为 90 等信息。

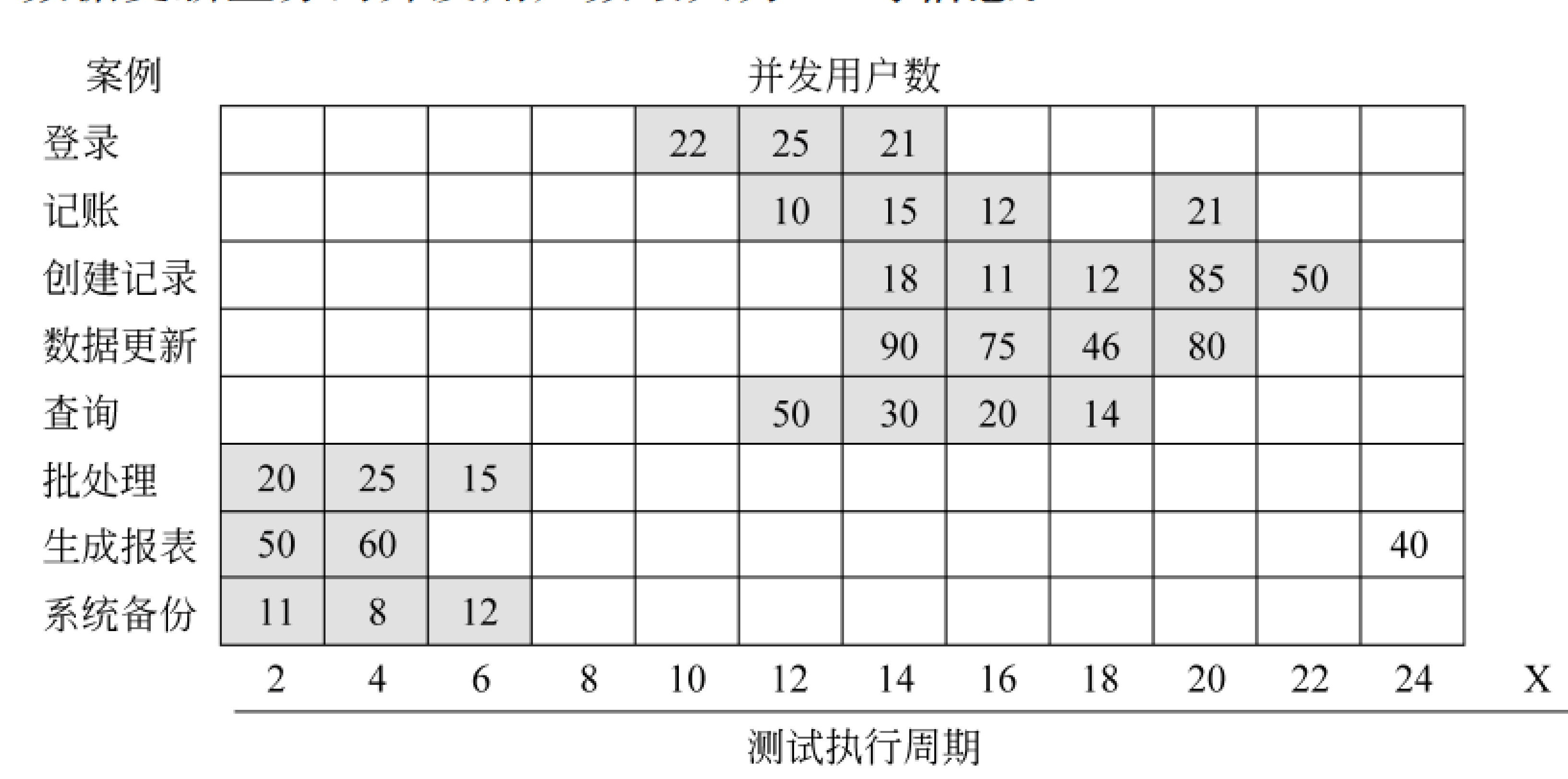


图 13-1　任务分布图

(2) 交易混合图方法。

使用交易混合图方法应关注以下三点:

① 系统日常业务主要有哪些操作,高峰期主要有哪些操作。

② 数据库操作有多少。

③ 如果任务失败,那么商业风险有多少。

选择重点交易的指标为高吞吐量、高数据库 I/O、高商业风险。举例如表 13-2 所示,“登录”、“生成订单”以及“发货”为性能测试重点。

表 13-2　交易混合表

交易名称	日常业务/hr	高峰期业务/hr	Web 服务器负载	数据库服务器负载	风险
登录	70	210	高	低	大
开一个新账号	10	15	中等	中等	小
生成订单	130	180	中等	中等	中
更新订单	20	30	中等	中等	大
发货	40	90	中等	高	大

(3) 用户概况图方法。

使用用户概况图方法应关注以下两点:

① 哪些任务是每个用户都要执行的。

② 针对每个用户,不同人物的比例如何。

如表 13-3 所示为任务频率表,性能测试需要模拟不同用户角色的压力。

表 13-3　任务频率表

	订票部门(170)	飞行部门(50)	经理(30)
输入订单	100	25	
更新订单	50	10	
计算飞行里程		70	5
计算销售			8

13.3.2　测试环境

测试环境直接影响测试效果,所有的测试结果都是在一定软硬件环境约束下的结果,测试环境不同,测试结果可能会有所不同,对于性能测试更是如此,因为性能测试结果往往是一组和时间有关的值,因此对性能测试环境的准备就显得特别重要。

(1) 测试环境的基本原则。

① 要满足软件运行的最低要求,不一定选择将要部署的环境;

② 选用与被测系统相一致的操作系统和软件平台;

③ 营造相对独立的测试环境;

④ 无毒的环境。

(2) 性能测试的测试环境。

性能测试环境准备过程中可以参考测试环境的基本准则,但是又要考虑性能测试的特殊性和性能测试目的。性能测试一般强调“真实”应用环境下的性能表现,从而实现性能评估、故障定位以及性能优化的目的,因此在性能测试环境的准备中要注意以下几点:

① 如果是完全真实的应用运行环境,要尽可能降低测试对现有业务的影响;

② 如果是建立近似的真实环境,要首先达到服务器、数据库以及中间件的真实,并且要具备一定的数据量,客户端可以次要考虑;

③ 必须考虑测试工具的硬件和软件配置要求;

④ 配置与业务相关联的测试环境需求;

⑤ 测试环境中应包括对交互操作的支持;

⑥ 测试环境中应该包括安装、备份及恢复过程。

(3) 测试环境配置。

① 操作系统的版本(包括各种服务、安装及修改补丁);

② 网络软件的版本;

③ 传输协议;

④ 服务器及工作站机器;

⑤ 测试工具配置。

(4) 良好的测试环境标准。

① 保证达到测试执行的技术需求;

② 保证得到稳定的、可重复的、正确的测试结果。

大多数情况下强调在真实环境下检测系统性能，在实施过程中大家认为这样做会遇到很多阻力，如真实环境下，不允许性能测试为系统带来大量的垃圾数据；测试数据与真实业务数据混在一起无法控制测试结果；性能测试如果使服务器宕机，会给系统带来巨大的损失等。那么应该如何理解“在真实环境下检测系统性能”呢？

在性能测试中强调的“真实环境”是指后台服务器与客户端应用要与实际应用环境保持一致，同时，这里也包括了与业务有关的软硬件配置环境和数据量环境等，可以看出将网络环境排除在外。原因是网络环境缓解了客户端对服务器所造成的并发负载压力，网络规模越大、网络类型越多、网络拓扑越复杂、网络流量越纷繁交织，对客户端的并发负载压力缓解程度越大。

13.3.3 数据准备

实施性能测试时，需要运行系统相关业务，这时需要一些数据支持才可运行业务，这部分数据即为初始测试数据。例如 ERP 软件运行前财务账套的准备。

在初始的测试环境中需要输入一些适当的测试数据，目的是识别数据状态并且验证用于测试的测试案例。在正式的测试开始以前对测试案例进行调试，将正式测试开始时的错误降到最低。在测试进行到关键过程领域时，非常有必要进行数据状态的备份。制造初始数据意味着将合适的数据存储下来，需要的时候恢复它，初始数据提供了一个基线用来评估测试执行结果。

对系统实施性能测试的时候，经常会需要准备大数据量、实施独立的测试，或者与并发负载压力相结合的性能测试，这部分数据为业务测试数据。例如飞机订票系统查询订票信息，就需要准备大量的订票记录。又如测试并发查询业务，那么要求对应的数据库和表中有相当的数据量，以及数据的种类应能覆盖全部业务。

在性能测试过程中，为了模拟不同的虚拟用户的真实负载，需要将一部分业务数据参数化，这部分数据为参数化测试数据。例如模拟不同的用户登录系统，就需要准备大量用户名及相应密码参数数据。

还需要考虑特殊系统需要的测试数据，模拟真实环境测试。有些软件特别是面向大众的商品化软件，在测试时常常需要考察在真实环境中的表现，如测试杀毒软件的扫描速度时，硬盘上布置的不同类型文件的比例要尽量接近真实环境，这样测试出来的数据才有实际意义。

系统所需数据的分析可以参考以下方式：

(1) 历史数据分析有助于数据量级的确定。从历史数据入手，找出高峰期数据量。

(2) 从其他相似或者相同的系统入手，进行数据分析，找出高峰期数据量。

(3) 无历史或者相关系统可以参考的时候，就要对系统的性能数据进行估算，包含系统容量，并发数等数据，估算及给相关人员进行评审或者修订以后，按照大家同意的性能指标进行测试。

测试数据最好和真实数据相同，如果能够获得真实系统运行三个月的数据，就可以在

此基础上进行性能测试。测试数据最重要的是要达到真实环境运行下的数据量级。

如何解决“大数据量测试需求,但很难在较短的时间内生成大量业务数据”?

(1) 可以借助自动化测试工具,利用数据库测试数据自动生成工具,例如TESTBytes,确定需要生成的数据类型,通过与数据库的连接来自动生成数百万运行正确的测试数据。

(2) 利用自动化负载压力测试工具,模拟用户业务操作,同时并发数百个或者数千个用户生成相关数据,并且测试工程师并不需要清楚地知道数据表与表之间的关系等细节内容,这样就事半功倍了。例如要生成订单,不必考虑订单中的信息在数据库内部到底与哪些表有关系,只需要简单录制一个用户生成订单的操作,然后模拟大量虚拟用户生成订单数据就可以了。

(3) 还可以针对某个应用,在了解整个数据库结构的基础上,自主开发数据生成工具,也可以利用数据库本身提供的辅助工具来生成数据。

13.3.4 测试策略

制定测试策略时,应针对用户的需求给出解决对策,即采用什么样的方法,来实现用户测试的目的。

对于一个特定的业务系统,用户一般会分散在一天的各个时间段进行访问。在不同的时间段中,用户使用业务系统的频率不同,而系统的繁忙程度不同。在一些特定的条件下,可能出现短时间内用户集中访问某个业务系统的情况。例如对于公文处理系统而言,可能就存在短时间内大量用户查看并办理某条公文的情况。在进行性能测试时,应当使用“考虑最坏情况的原则”,也就是应当在用户使用业务系统最频繁、对系统造成最大压力的情况下对系统的功能进行测试,判断各功能和页面是否能够满足性能的要求,系统的响应时间是否过长。

另一方面,系统性能的验证必须做到“覆盖全面”。虽然系统中各个功能的使用频率并不相同,一些功能的使用频率相对于其他功能来说比较低,但是在进行性能测试和优化时,不能忽略这些功能,编制测试用例时也不能仅仅选择最常用功能。例如可能所有的用户都会访问通知列表,但是一般只有5%的用户会使用通过系统设置模块查找某个用户的信息;但是在测试时,并不能因为查看用户信息功能的使用频率相对较少,而忽略掉这项功能的测试。所以,进行系统性能测试时,对于不同业务,用户的访问比例应该做一个合理分配。

在测试策略上还应该考虑,同一个系统在不同硬件环境下的性能表现,从而让系统在满足需求的情况下,硬件配置也能达到一个最佳的状态。过分地增加硬件来满足需求也是一种浪费,况且增加硬件设备不是能解决所有性能问题的。

通常软件系统的性能与测试要素(用户数、测试功能、用户分布、数据量、硬件环境、软件环境)密切相关。对同一个软件系统,当测试要素中有一个变化时,其性能表现也会发生显著的变化。通过对测试要素的值或变化进行组合,来达到不同的目的,如相同数据量等条件下用户数不断变化,来验证软件系统所能支持的并发数。

不同的测试目的，对应的测试策略不同，采用的性能测试方法或类型也不同。

(1) 性能符合性验证：负载测试、疲劳强度测试；

(2) 性能能力验证：压力测试、疲劳强度测试；

(3) 性能调优：负载测试、疲劳强度测试。

负载测试主要指常规的性能测试，通过模拟生产运行的业务压力和使用场景组合来测试系统的性能是否满足生产要求。各测试要素都是相对固定的。

压力测试是对系统不断施加压力的测试，是通过确定一个系统的瓶颈或者不能接收用户请求的性能点，来获得系统能提供的最大服务级别的测试。通俗地讲，压力测试是为了发现在什么条件下应用程序的性能会变得不可接受。部分测试要素是变化的。但在测试前，需要确定性能不可接受的标准。

疲劳强度测试通常是采用系统稳定运行情况下能够支持的最大并发用户数或者日常运行用户数，持续执行一段时间业务，通过综合分析业务执行指标和资源监控来确定系统处理最大工作量强度性能的过程。测试要素相对固定。

13.3.5 人力与时间安排

最后一条，就是要根据项目的进度要求以及规模，来进行人力与时间的安排。对于大型的性能测试，项目前期的需求调研，环境的部署，工具的选购或开发，人员对测试工具的学习与使用，性能测试的进行，后期数据的分析与调优，都需要人员安排。有的时候可能还需要系统工程师、数据库工程师、软件开发工程师、网络工程师以及性能测试工程师共同参与配合，并不是一个性能测试人员就可以全部完成的。

如何把控性能测试的同步进行，后期测试数据的汇总与分析，是一个非常复杂的过程。有些大规模的项目，前期的准备工作就需要几个月的时间。因此对于大项目的性能测试，人员与时间安排也至关重要。

13.4 业务流程

通过之前对系统进行的性能测试需求分析和性能测试计划的制订，已经了解到系统所需的数据和运行环境，以及用户一般的操作习惯。接下来就可以对系统的业务流程进行分析、录制了。

13.4.1 业务流程介绍与案例

所谓业务流程，就是在一个应用内有一组用户的操作或者一组步骤执行去实现一个任务，如订票业务，修改确认等。只有明确了系统的业务流程，才能模拟出真实的系统使用情况，进而找到今后可能发生的缺陷。

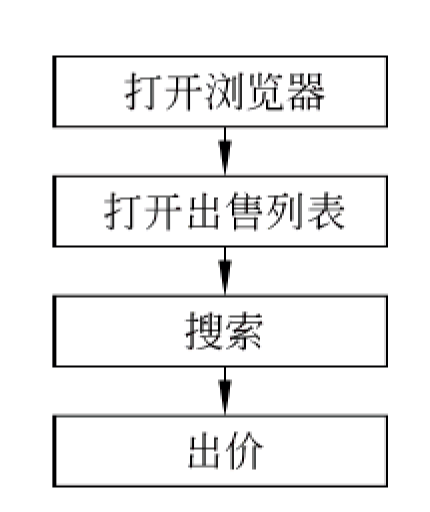


图 13-2　一个典型的拍卖业务流程

对于一个典型的拍卖案例而言，业务流程如图 13-2

所示。

不过对于一些网站而言，可能有多个业务流程，如搜索、登录、留言等。

13.4.2 业务流程分析

在确定了待测试系统的业务流程之后，就要对这些业务进行分析。

首先要明确业务流程中用户的操作步骤和可能输入的数据。这既可以让人们了解业务操作步骤，又能让人们了解哪些地方需要使用动态的测试数据。

如表13-4所示的是一般系统中普遍都有的登录业务。这张表展示了登录的每一个步骤。同时可以看出，表中的mercury和test是登录业务所需的动态测试数据。

表13-4 登录业务操作流程

登录业务操作步骤	输入数据
1. 浏览登录页面	
2. 输入用户名和密码	用户名：mercury 密码：test
3. 单击登录按钮	
4. 等待确认	

性能测试的过程在很大程度上依赖于根据实际操作所录制的脚本。虽然在做性能测试的时候要兼顾每个功能模块，但对于一些用户使用频率较低，访问量较少的业务，是没有必要全部录制下来进行大规模测试的。

那么什么样的业务是需要录制的？

(1) 核心业务。

对于一个系统而言，核心业务是必须能在任何情况下都正常运行，绝对不能出现问题的。例如：邮箱系统中的收发邮件业务，电子商务系统中的交易过程。这部分业务对性能的要求极高，一旦出现问题，造成的损失也是不可估量的。

要想确定系统中的核心业务，可以与应用专家、产品经理、管理组等相关人员进行沟通。

(2) 高吞吐量的业务。

用户访问量在系统中并不是均匀分布的。类似访问主页、搜索以及用户登录这样的业务在一个系统的总业务量中可以占到90%甚至以上的比例。这部分业务也是性能测试的重点。如果系统中最常用的业务都无法保证高负载下的正常运转，那么用户也许就会放弃使用该系统，从而造成巨大的损失。

(3) 包含动态内容的业务。

包含动态内容的业务在性能测试中也是需要录制下来的。例如，在一些系统的订单确认、密码修改、账号激活以及登录等业务中，都会由系统自动生成动态的确认号或验证码。此外，根据用户的操作和权限，系统也会生成动态页面。包括音频、视频这些流媒体的播放，都属于包含动态内容的业务。

表 13-5 是对一个具体的订票系统的业务流程的分析。

表 13-5 订票业务流程分析

业务流程名	正常时段吞吐量	高峰时段吞吐量	包含的动态数据量	核心级别	是否需要录制
登录	70/hr	210/hr	少量	非常核心	必须录制
创建新账号	10/hr	15/hr	中等	不太核心	可选录制
搜索飞机票	130/hr	180/hr	中等	中等	可选录制
查看订单	20/hr	30/hr	中等	非常核心	必须录制
订票	40/hr	90/hr	大量	非常核心	必须录制

13.5 步骤测量

在业务流程中，对于不同类型的业务，用户可能有不同的预期响应时间和不可接受的响应时间，如表 13-6 所示。这些时间可以通过与应用专家和管理组的沟通得到。

表 13-6 预期时间和不可接受的响应时间

业务流程	预期时间	不可接受的响应时间
查看机票订单	8～12s	20s
搜索航班	6～8s	12s
订票	12～20s	30s

但是实际录制业务流程的时候，有些业务并不适合，或者根本无法单独录制，必须与其他业务一起。如图 13-3 所示是一个完整的航空公司订票业务流程。

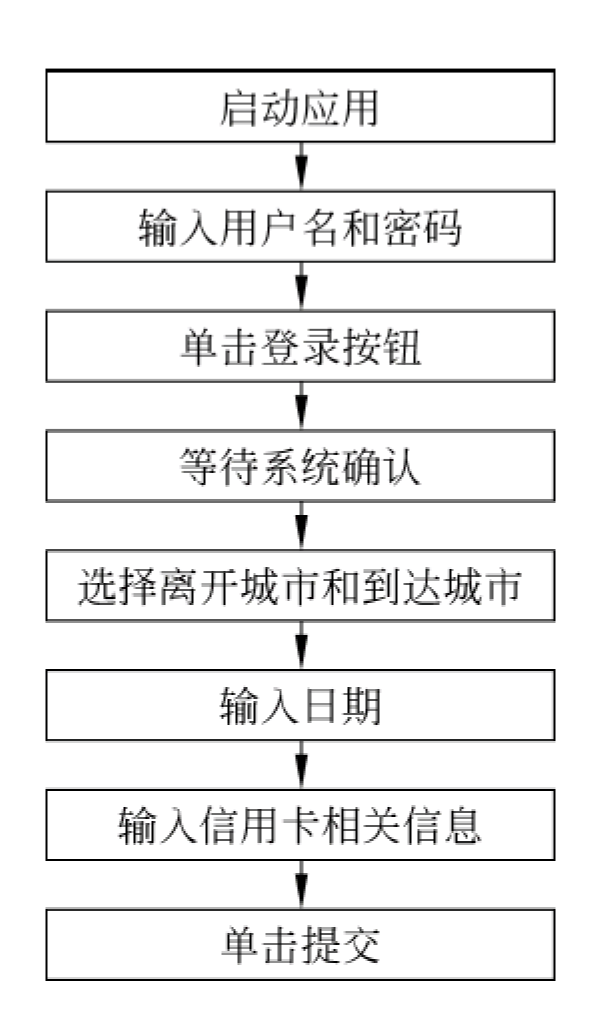


图 13-3 一个完整的航空公司订票业务流程

很显然，其中的订票流程必须在已登录的情况下才能完成。也就是说，在录制订票业务时，必然会将登录业务也录制进来。这种情况非常普遍，例如用户查看个人信息、收发邮件、网上交易等。这就带来了一个问题：如果想从业务流程中获取登录的响应时间，或是获取提交订单的响应时间，要如何做呢？

在这里，引入“事务”这一概念。所谓“事务”，就是在业务流程中衡量一个或者多个步骤的执行时间的方法，通过事务来测量业务流程中步骤耗费的时间。

在图 13-3 中，如果想获取登录的响应时间，可以将“单击登录按钮”和“等待系统确认”当作一个业务，也就是从单击按钮直到页面显示登录成功的页面。这样就可

以准确地得到登录的响应时间。

13.6 本章小结

本章介绍了性能测试中需求分析的过程。性能测试需求要保证所有确定的性能测试需求内容都是可以用某种方法来明确判断是否符合软件需求文档或其他重要文档中的描述的。性能测试需求分析是性能测试的先决条件,也将耗费测试人员大量的时间。所以,一定要重视性能测试需求分析,精准把握测试所需内容、条件,为性能测试的成功打下坚实的根基。

第 14 章　测试脚本编写

在整个性能测试的过程中，测试脚本的编写无疑是十分重要的一环。因为只有通过测试脚本的实际运行，才能实现先前制订的测试方案，才能对整个被测系统的状况有一个全面而深入的了解。可以说，测试脚本编写的好坏程度直接关系到测试结果的准确性，以及测试过程的执行效率。

LoadRunner 生成脚本的方式有两种，一种是自己编写手动添加或嵌入源代码；一种是通过 LoadRunner 提供的录制功能，运行程序自动录制生成脚本。这两种方式各有利弊，但首选还是录制生成脚本，因为它简单且智能化，对于测试初学者来说更加容易操作。但是仅靠着自动录制脚本，可能无法满足用户的复杂要求，这就需要手工添加函数，进行必要的手动关联或在函数中进行参数化来配合，增强脚本的实用性。手写添加增强脚本的独特之处在于：

（1）可读性好，流程清晰，检查点截取含义明确。业务级的代码读起来总比协议级代码更容易让人理解，也更容易维护，而且必要时可建立一个脚本库。而录制生成的代码大多没有维护的价值，现炒现卖。

（2）手写脚本比录制脚本更能真实地模拟应用运行。因为录制的脚本是截获了网络包，生成的协议级代码，而略掉了客户端的处理逻辑。

（3）手写脚本比录制脚本更能提高测试人员的技术水平。LoadRunner 提供了 Java user、VB user、C user 等语言类型的脚本，允许用户根据不同的测试要求自定义开发各种语言类型的测试脚本。

增强脚本的好坏关系到这个脚本是否能在实际运行环境中更真实地进行模拟操作。

至于具体使用哪种方式来生成脚本，还应该以脚本模拟程序的真实有效为准。例如，有些程序只需要执行迭代多次操作，没有特殊要求，选择自动生成的脚本就可以了；有些程序需要加入参数化方可满足用户的要求，此时应该使用增强的手工脚本。再就是结合项目进度、开发难易程度等因素考虑。

本章将以 LoadRunner 为例，从参数化脚本，手工关联，自动关联，日志高级应用，以及高级脚本技术几个方面来阐述测试脚本的编写方法和注意事项。

14.1　参数化脚本

14.1.1　参数化的目的

参数化脚本和非参数化脚本的区别在于，如图 14-1 所示，非参数化的录制脚本每次运行是一个硬编码的脚本，其对系统的输入是固定的，如图例，每次仅能选择固定的 San Francisco。而参数化脚本则可以动态地选择输入参数，如图例，每次都可以选择任意的城市。

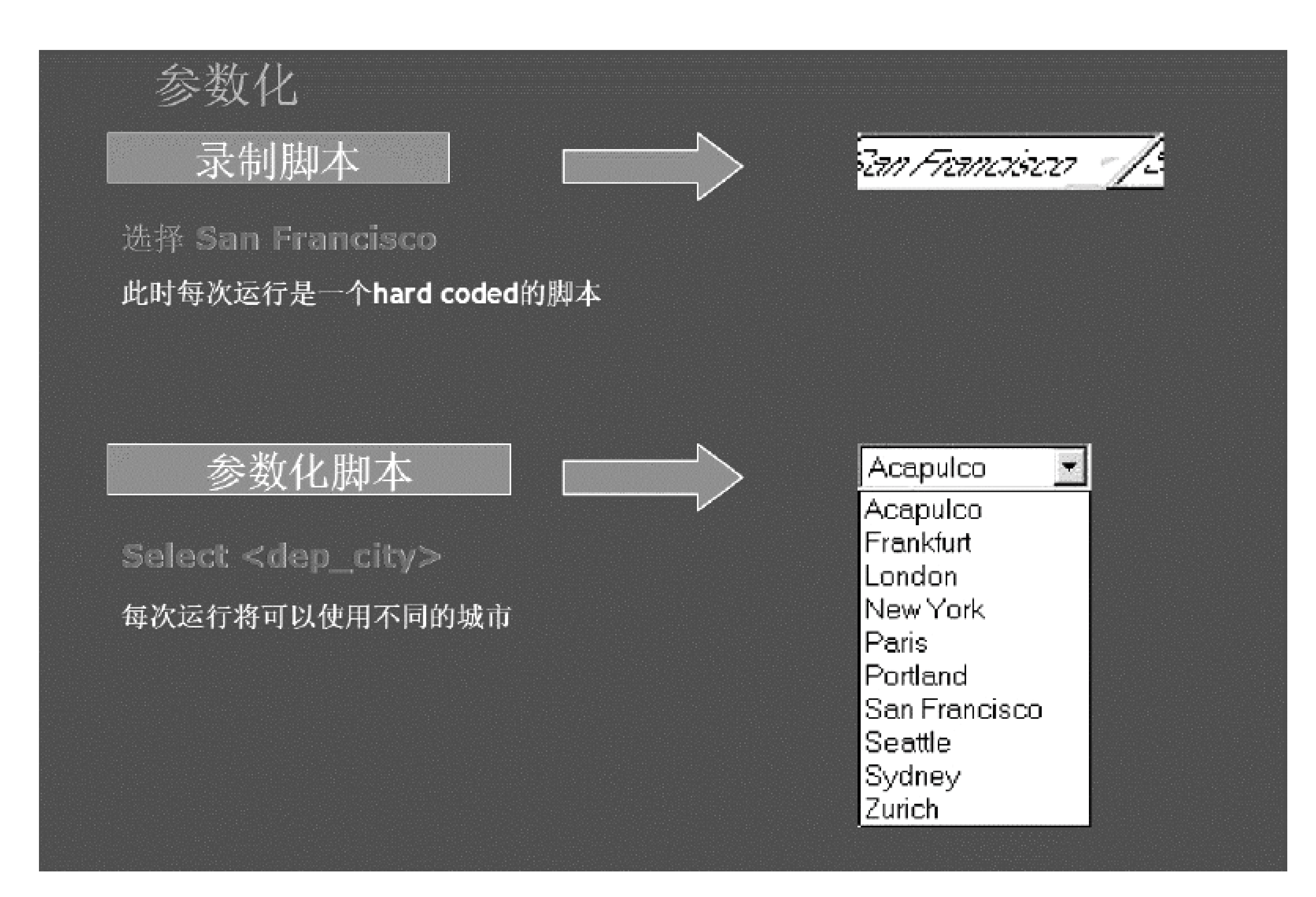


图 14-1　非参数化录制脚本和参数化脚本

参数化有两个目的，其一是避免缓存问题，缓存会造成响应的时间不一致，其二是要模拟真实用户的使用情况。

如图 14-2 所示，真实的用户在会在应用中使用不同的数据，而简单录制的脚本（虚拟用户）则是一个静态数据脚本。

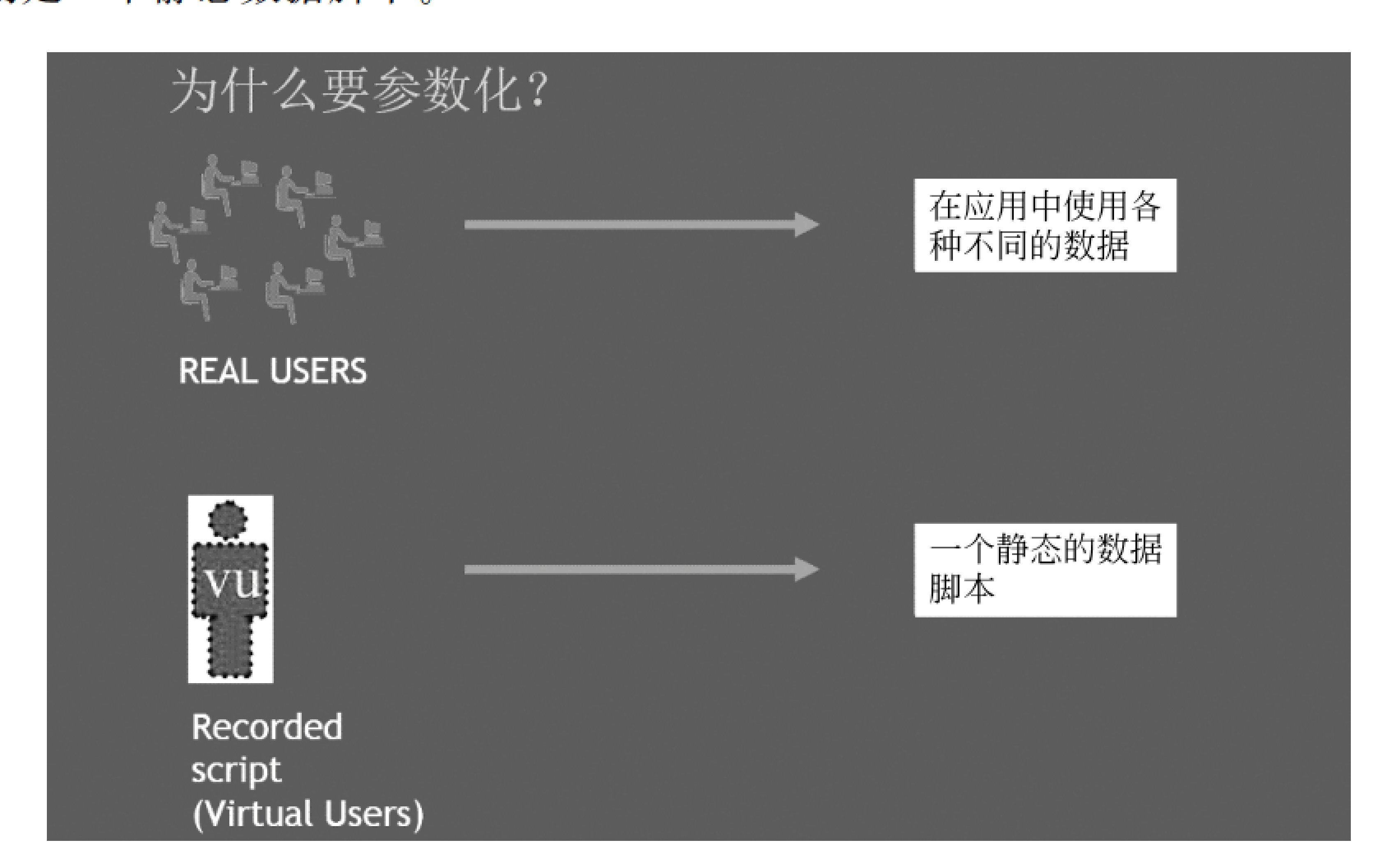


图 14-2　实际用户和录制脚本（虚拟用户）

在实际使用的被测系统中，往往会用缓存来提升系统的响应时间。如果不进行参数化，那么对单一的静态数据的访问，必然会被系统置于缓存中，那么后续的访问将几乎永远可以直接从系统缓存中得到想要的结果，而实际上，缓存是有限的，如果大规模的真实用户访问系统，其需求的结果必然不能完全通过缓存中的数据来满足。这样的情况下，如

果不对系统进行参数化，将使得系统的响应速度远远快于实际使用中的表现，这样的结果显然不是人们想要的。另外，真实用户的操作往往带有很大的随意性，有些用户对系统熟悉，则其操作可能较为迅速，另外一些则有可能较为缓慢；每一个用户进行违法操作的概率也不尽相同，等等。这些都使得简单的非参数化脚本，不能很好地模拟真实用户的行为。

因此，要对脚本进行参数化，其一是避免缓存问题，因为缓存会造成响应时间不一致，其二是要模拟真实用户的使用情况。实际上第一点也可以包含在第二点中。最根本的目的就是要模拟真实用户的使用情况，以尽可能估算出系统的实际性能表现。

14.1.2 什么时候进行参数化

说完参数化脚本的目的，现在结合如图 14-3 所示的购票过程来谈谈什么时候进行参数化，以及为什么要在这些时候进行参数化。

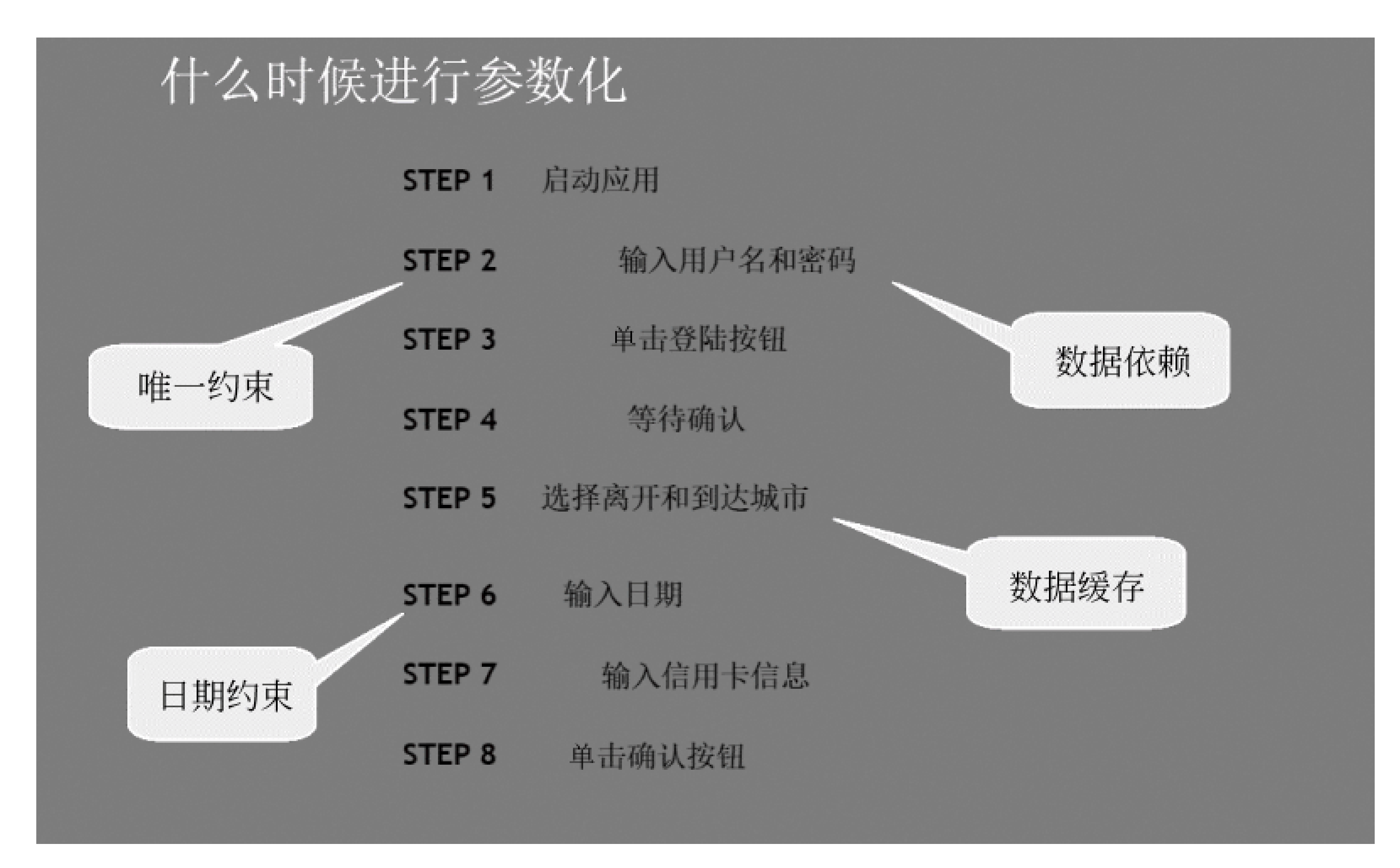


图 14-3 购票过程

如图 14-3 所示，在 STEP2 输入用户名和密码，STEP5 选择离开和到达城市，STEP6 输入日期时，需要进行参数化。

具体来说，会在具有唯一约束，有数据依赖，有数据缓存或者有日期约束的时候进行参数化。

以上图中的购票过程为例，STEP2 中输入的用户名具有唯一性，其在不同的实际用户间肯定是不同的，因此也只能被一个虚拟用户所使用，所以需要对其进行参数化。同是在这一步，用户的密码是依赖于用户名的，两者存在数据依赖关系，因此不可能用固定的数据或者虽不固定实际上却和用户名不相关的数据进行测试，而必须进行参数化。

在 STEP5 选择离开和到达的城市时，因为存在如图 14-4 所示的缓存情况，在不进行参数化的情况下会导致系统测试的响应速度快于系统实际的相应速度，因此，需要对其进行参数化。如图 14-4 所示，上方不参数化的脚本其平均响应时间仅为 2.5s，而参数化的

脚本的平均响应时间却是 4.6s。因此参数化无疑更加符合用户实际的使用情况。

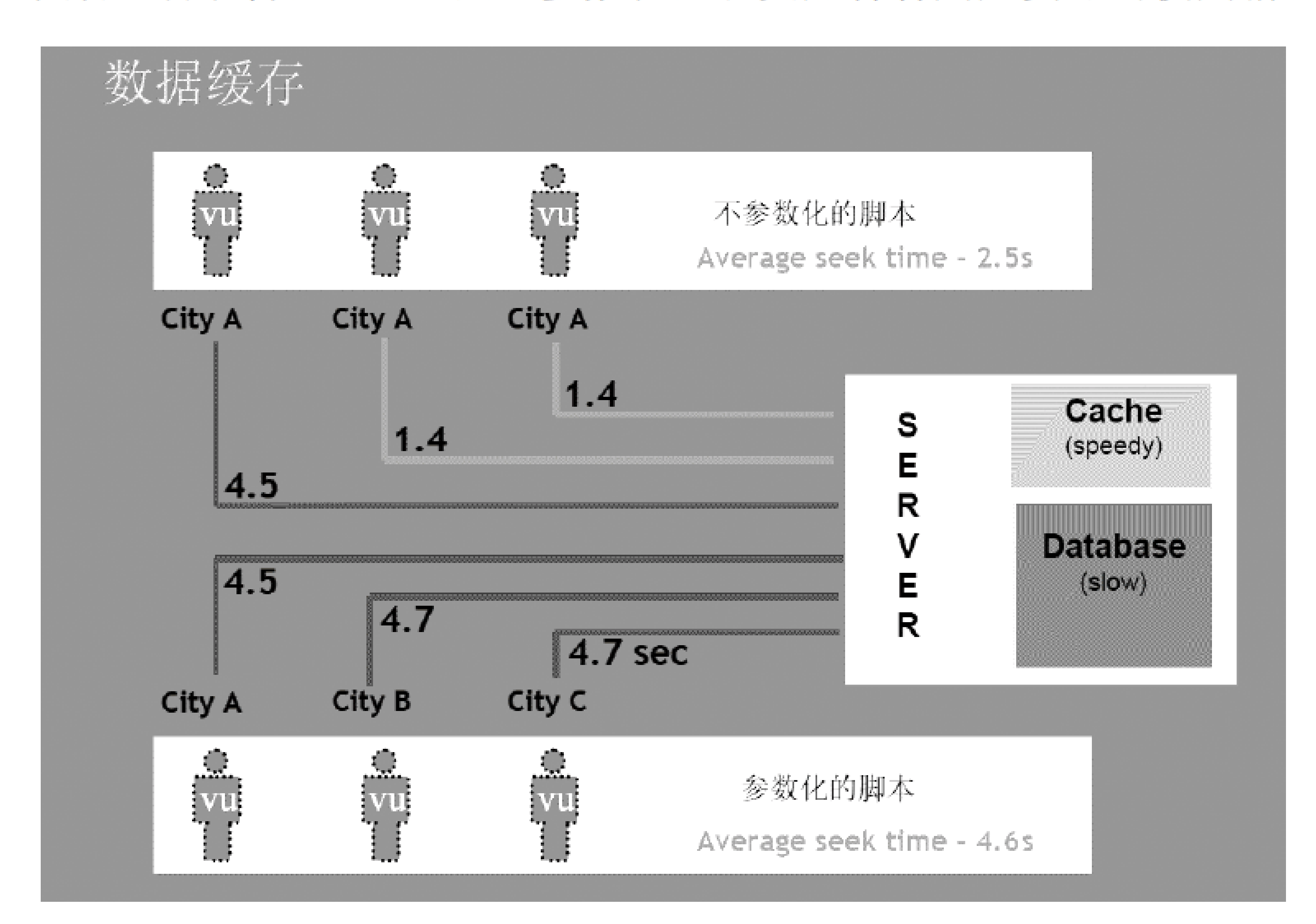


图 14-4 存在数据缓存的情况

在 STEP6 输入离开的日期时，所输入的日期仅能是当前或者当前日期以后，而不能是当前日期之前。因此必须按照这一规则对其进行参数化。

因为只有在这些时候进行参数化，方能保证虚拟用户能和实际用户的操作方式最大限度地吻合，因此需要在具有唯一约束，有数据依赖，有数据缓存或者有日期约束的时候进行参数化。

14.1.3 怎样参数化输入数据

前面讲了参数化的目的和什么时候进行参数化，现在说一说该怎样参数化输入数据这个问题，一般采取以下的步骤：

(1) 决定哪些字段需要参数化。

(2) 用参数化值代替原有的值。

(3) 决定参数类型。

(4) 创建数据文件。

(5) 选择数据访问方式以及运行类型。

(6) 运行脚本并且检验参数设置的正确性。

关于哪些字段需要参数化的问题，已经在 14.1.2 节中详细交代，此处不作赘述。在确定了需要参数化的字段以后，要用参数化值代替原有的值。接着需要决定参数的类型。常用的数据类型如表 14-1 所示。

表 14-1 常用的数据类型

参 数 类 型	说　　明
Date/Time	在需要输入日期、时间的地方，可以用 Date/Time 来代替，可以选择日期格式，也可进行定制
Group Name	该类型的参数用执行脚本的 VU 所属组的名称来替代。但是在 VuGen 中运行时，该值为 None
LoadGenerator Name	LoadRunner 使用该虚拟用户所在的 Load Generator 机器名来代替参数
Iteration Number	LoadRunner 使用该测试脚本当前循环的次数来代替参数
Random Number	随机数，可以设置产生随机数的范围
Unique Number	唯一值来代替参数
Vuser ID	LoadRunner 使用该虚拟用户的 ID 来代替参数值，该 ID 是由 Controller 来控制的，在 VuGen 中运行脚本时，该值为－1
File/Table	可以在属性设置中编辑文件，添加内容，也可以从数据库中提取数据
UseDefinedFunction	从 dll 的简单函数中获取信息替代参数

当参数类型为 File 时还需要同时创建数据文件并指明数据源。参数的数据来源通常有三种：主数据，自定义用户数据以及动态数据。其中主数据来源于各种被测系统内的数据库，例如存在的用户名或者密码以及相关的业务数据。自定义用户数据主要是一些用户的标识，例如用户的传真号、Email 地址。动态数据主要是应用执行之前根本无法预知的数据。

再下一步，选择数据访问方式以及运行类型。LoadRunner 中数据访问方式有以下几种：Sequential，Random，Unique，Same line as＜parameter＞。

对于 Sequential 方式，如表 14-2 所示，每个用户都是从同一行开始执行的。第一次循环执行第一行，第二次循环执行第二行。

表 14-2 Sequential 方式

Sequent	Vuser1	Vuser2	Vuser3
Iteration1	Acapulco	Acapulco	Acapulco
Iteration2	London	London	London
Iteration3	New York	New York	New York

对于 Random 方式，如表 14-3 所示，每次循环则随机获取参数，不保证数据唯一性。

表 14-3 Random 方法

Sequent	Vuser1	Vuser2	Vuser3
Iteration1	Paris	Seattle	London
Iteration2	London	NewYork	Zurich
Iteration3	Acapulco	Portland	Sydney

对于 Unique(by Row)的方式,如表 14-4 所示,每个用户都有自己的数据运行块,测试人员必须保证数据能够满足运行次数,以及数据的唯一性。

表 14-4　Unique 方式

Sequent	Vuser1	Vuser2	Vuser3
Iteration1	Acapulco	Paris	Seattle
Iteration2	London	Portland	Sydney
Iteration3	NewYork	San Francisco	Zurich

对于 Same Line as<parameter>方式,如表 14-5 所示,主要解决数据的依赖性问题,例如:用户名取为 jojo 时,密码就必须是 bean。

表 14-5　Same Line as<parameter>方式

Vuser	UserName	Password
Vuser1	jojo	bean
Vuser2	jno	rean
Vuser3	roro	bow

最后可以在 runtime Setting 中告诉参数池参数的执行次数,并选择 Extended Log 的参数输出属性来检查参数的正确性。

14.2　手工关联和自动关联

当录制脚本时,VuGen 会拦截 Client 端(浏览器)与 Server 端(网站服务器)之间的对话,并全部记录下来,产生脚本。

当执行脚本时,可以把 VuGen 想象成一个演员,它伪装成浏览器,然后根据脚本,把当初真的浏览器所说过的话,再对网站向服务器重新说一遍,企图骗过服务器,让服务器把网站内容传送给 VuGen。

记录在脚本中要跟服务器所说的话,完全与当初录制时所说的一样。这样的做法在遇到有些服务器时还是会失效的。这时就需要关联的做法骗过服务器。

所谓关联就是要把脚本中某些写死的数据,转变成撷取自服务器所送的、动态的、每次都不一样的数据。

以常见的登录系统为例。如图 14-5 所示,在登录后服务器会返回 SessionID,登录后的操作都必须要提交该 ID 确认身份。使用 VuGen 录制时,将会记录服务返回的 SessionID,并据此来进行下面的若干个请求。

待回放时,如图 14-6 所示,服务器会接收用户名和密码后返回一个新的 SessionID,而脚本依然发送旧的 SessionID 给服务器,最终因旧的 SessionID 已经失效造成脚本运行错误。因此要采用脚本关联机制,将原来固定的 SessionID 具体值用<session_id>参数代替,每次动态地获取相应的 SessionID 值,这样即可解决这一问题。

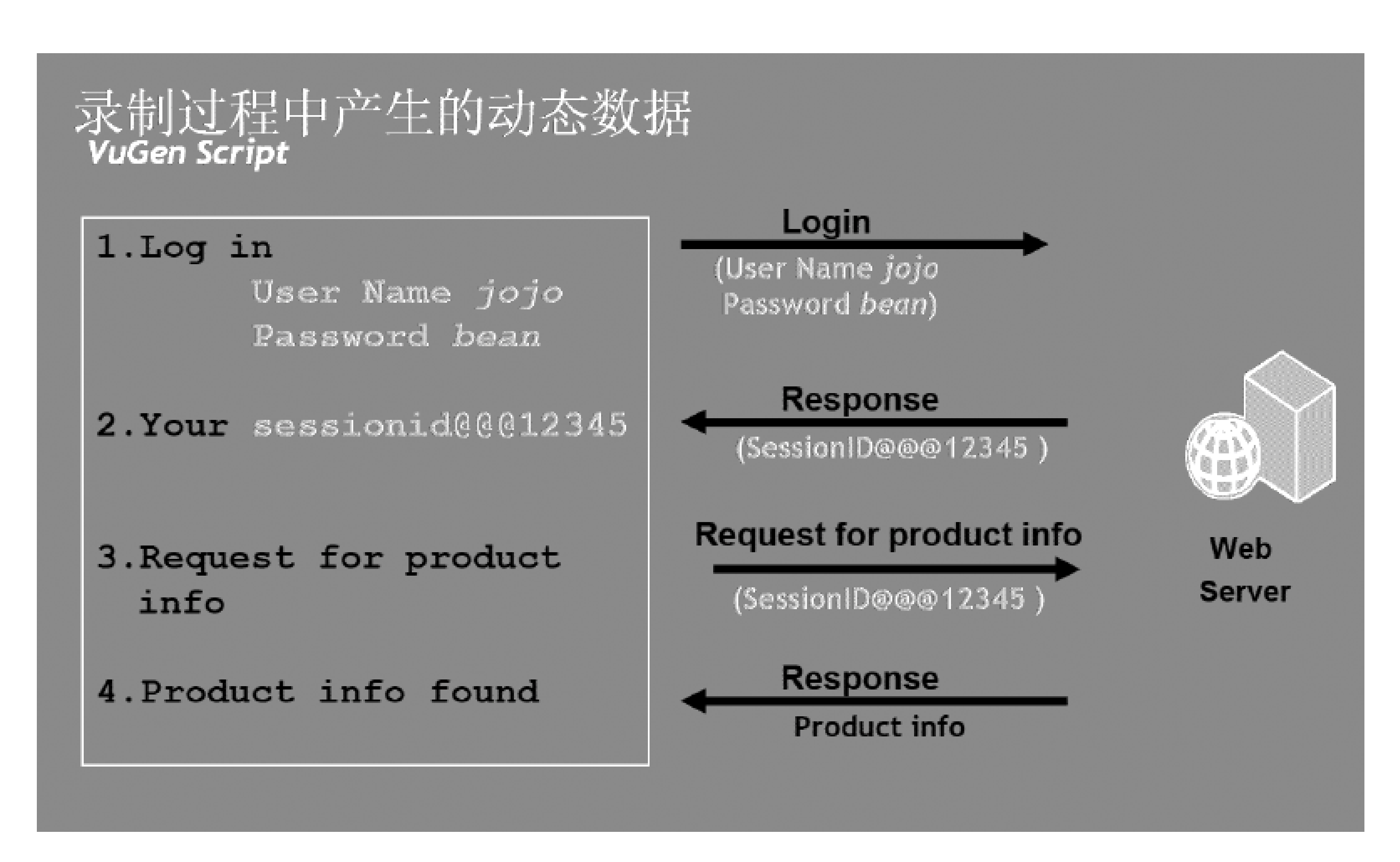


图 14-5 录制过程中产生的动态数据



图 14-6 回放过程中动态数据导致的问题

关联又分为手工关联和自动关联。

对于 LoadRunner 中的手工关联，其步骤如下：

(1) 在 runtime setting 中设置 Data returned by server 并回放脚本检查脚本，确定哪些脚本是因为关联问题导致失败的。

(2) 确定哪些数据需要关联。

(3) 找出动态数据的左右边界值以及出现的位置。

(4) 在脚本中添加 web_reg_save_param 函数。

(5) 在参数化脚本中的动态值。

（6）校验动态关联的正确性。

第(1)步执行后，LoadRunner 将能输出 Server 与 Client 的所有交互数据。接着需要回放脚本。首先查看哪步失败了，如图 14-7 所示。

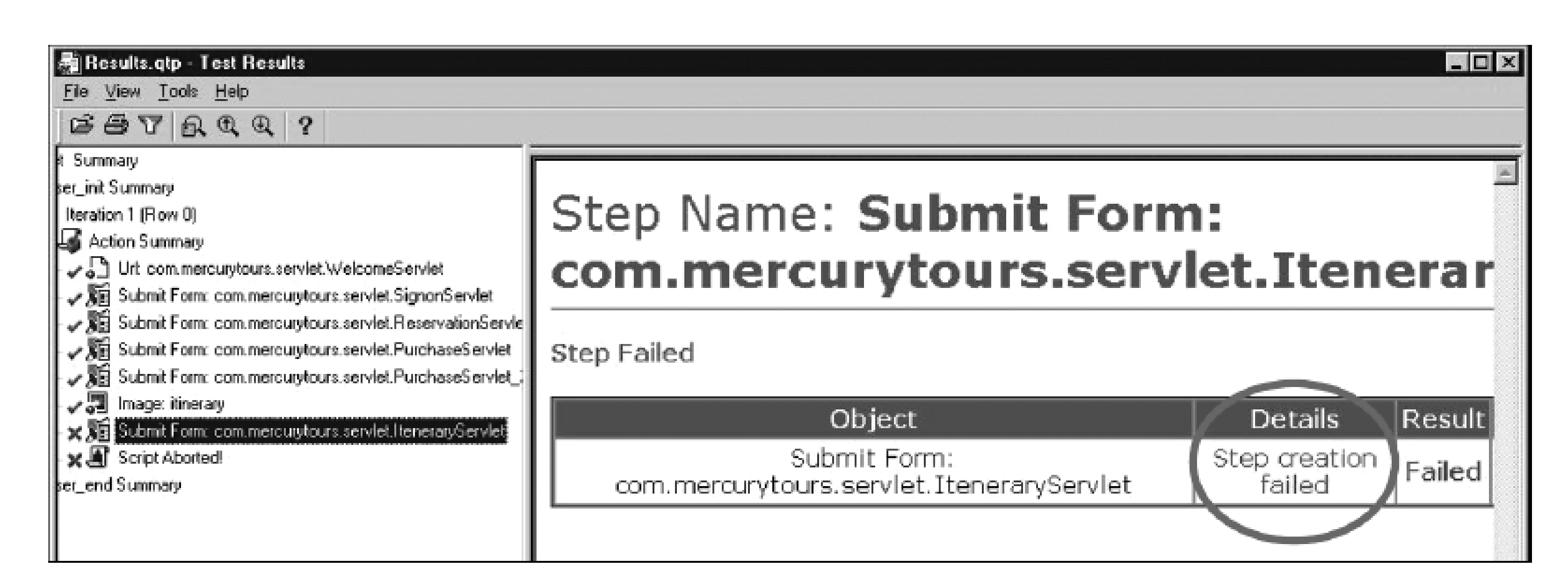


图 14-7　检查失败的步骤

然后要查看执行日志挖掘问题，如图 14-8 所示。

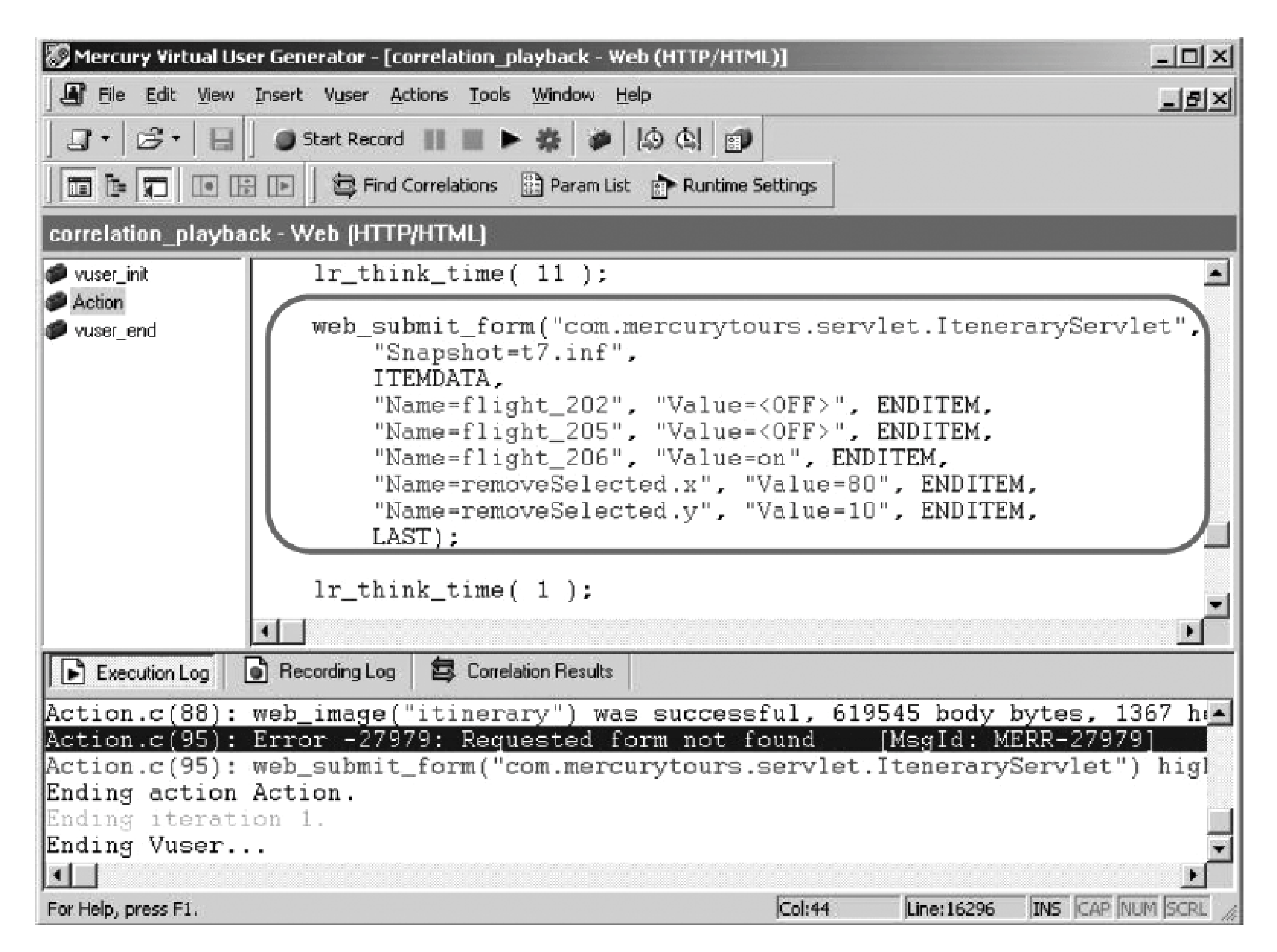


图 14-8　执行日志发现问题

执行日志中表现为 requested form was not found，在脚本中发现相关的测试行，原来是 delete itineray 信息没有找到。

接着要决定哪些参数需要关联，此时可借用 Wdiff 找出两次脚本的不同，帮助完成这一步。

在如图 14-9 所示的航班订票系统中，方框中标记的值表示鼠标在屏幕中的位置，这些值在脚本中会被使用，但不需要关联；延迟时间也在脚本中会被使用，也不需要关联，需要关联的就是航班号，因为其是动态的，需要关联。

由此可以看出需要关联的内容有两个特征。

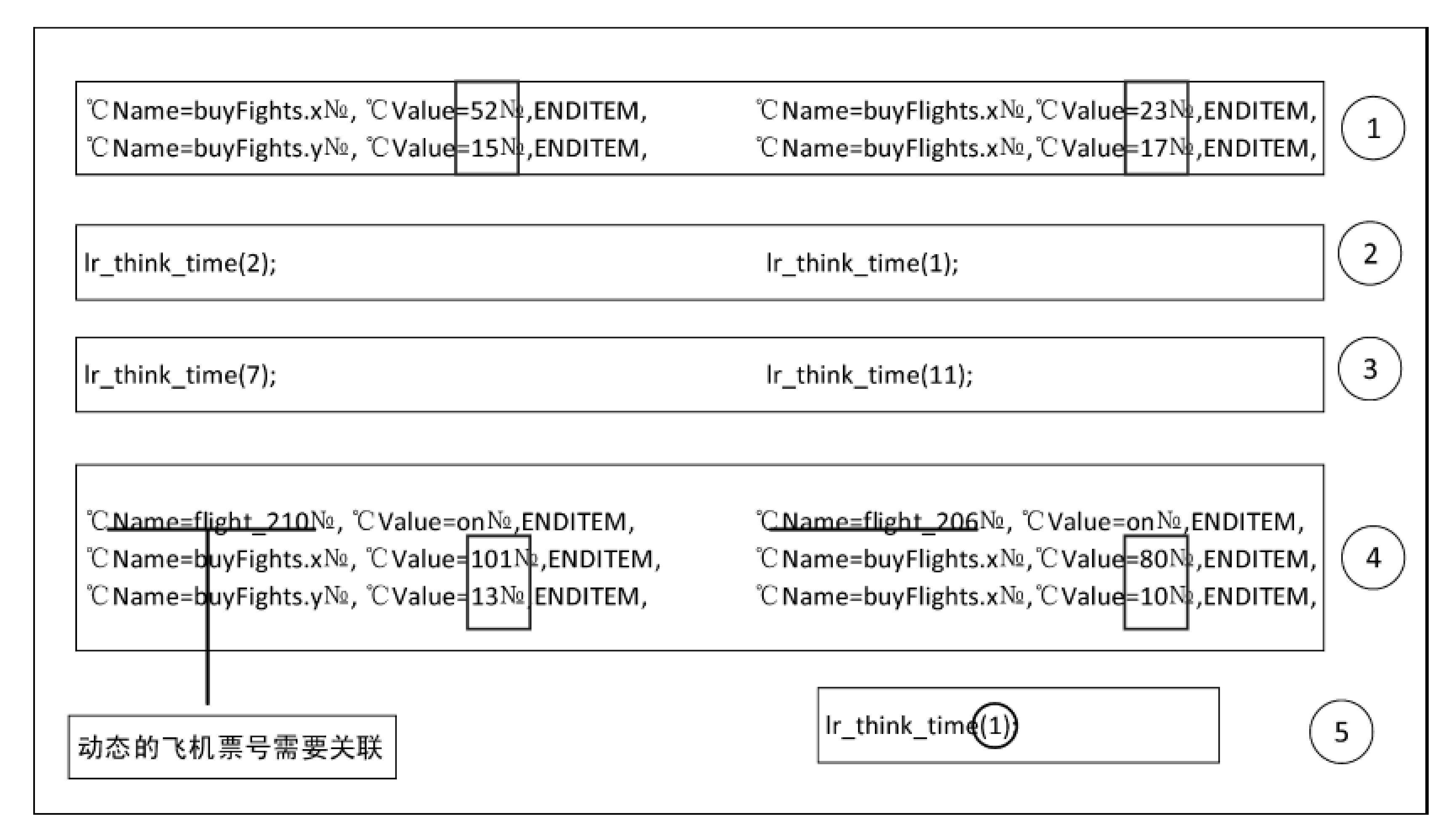


图 14-9　哪些值需要关联

（1）第一个特征：该动态内容一定首先是从服务器端产生并返回给客户端的。

（2）第二个特征：该客户端在得到该动态内容后一定会把它重新发送到服务器端。

只要满足这两个特征就需要对其进行关联。

再然后，需要找出动态数据的左右边界值以及出现位置，其具体位置包含：在什么文件下，在第几个左右边界值之后，如图 14-10 所示。

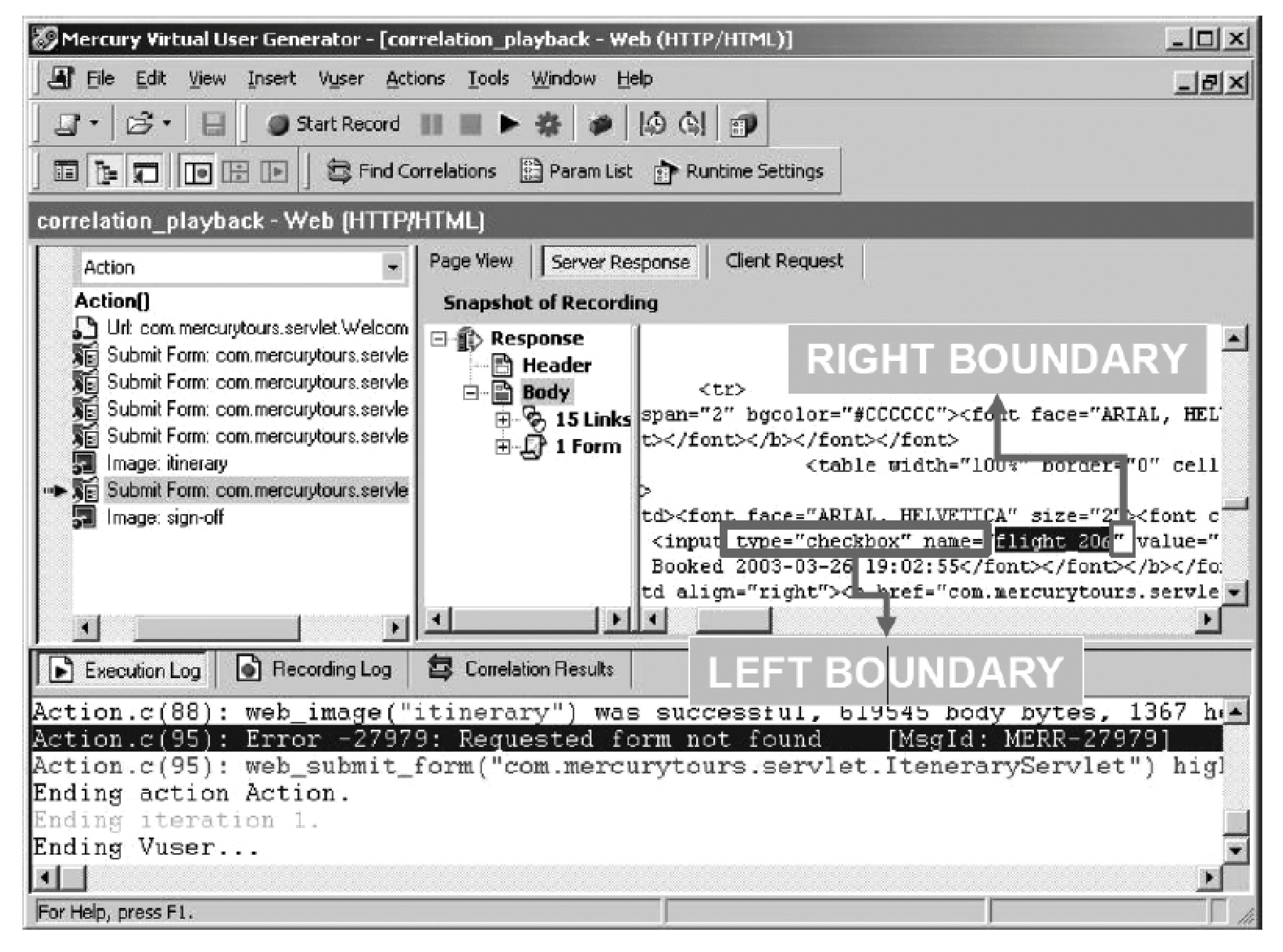


图 14-10　动态数据的边界值及其位置

需要将这些信息放入 web_reg_save_param 函数中，该函数的具体参数及其意义如图 14-11 所示。

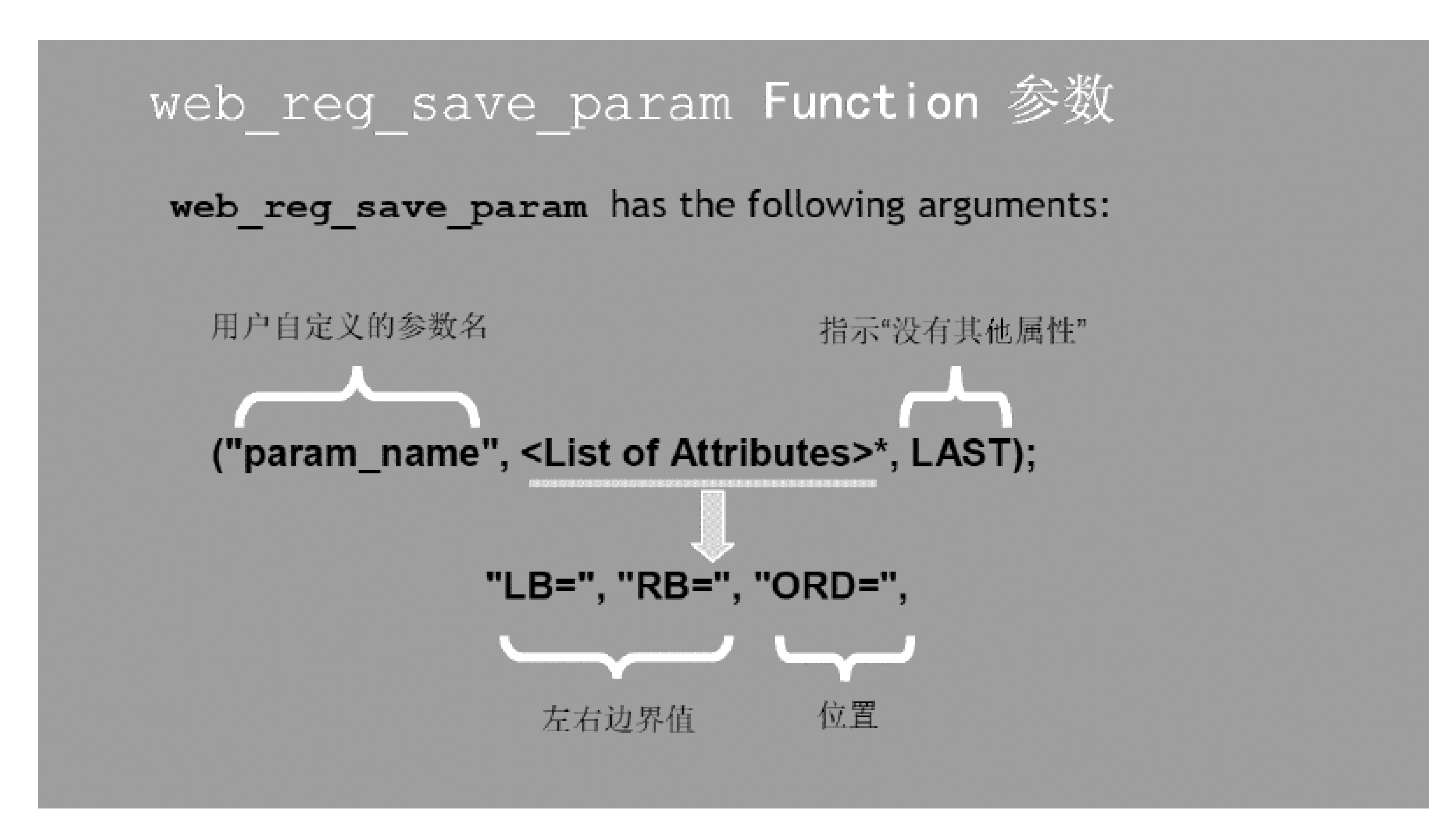


图 14-11　web_reg_save_param 函数的参数意义

最后将 web_reg_save_param 函数加入脚本中去，并校验执行结果即可。

对于自动关联，其原理在于，自动关联是 VuGen 提供的自动扫描关联处理策略，它的原理是对同一个脚本运行和录制时的服务器的返回进行比较，来自动查找变化部分，并且提示是否生成不关联。

在 LoadRunner 中可以先运行脚本一次，然后借用 Vuser 菜单中的 Scan Script for Correlations 选项以及 Correlate 按钮，来实现动态关联，此处不再赘述。

14.3　日志高级应用

当编译或者运行测试脚本时，难免会发生失败的情况。此时就要检查一下，在运行过程中输入的数据是什么？在运行过程中动态数据出现了什么问题？日志无疑是人们解决这些问题所必须要使用的功能，但是简单的日志功能又有可能不能完全满足人们的需要，此时，日志高级应用就显得十分必要了。日志高级应用的目的就在于从客户化的日志信息中挖掘执行中的错误。

一般来说，加高级日志的步骤如下：

(1) 初始化变量并设置为 0。

(2) 根据 check point 函数的 savecount 属性保存匹配的值。

(3) 把值保存到变量中。

(4) 校验变量值。

(5) 如果值为 0，将错误结果输入执行日志或者 Controller 的 output windows 中。

(6) 使用 lr_eval_string 和 lr_error_message 去写日志。

(7) 通过 return 函数去控制脚本运行。

下面以一个实际的 Action 来演示这一过程：

…

```
Action ()
{
第一步：    int itinerary_reserved=0;

    ……

    ……

    lr_start_transaction("purchase_ticket");

第二步：    web_reg_find("saveCount=booked_count", "Text=Your \n"
                "itinerary has been booked", LAST);

第三步：
web_submit_form("com.mercurytours.servlet.PurchaseServlet_2"………

第四步：    itinerary_reserved=atoi(lr_eval_string("{booked_count}"));

第五步：    if (itinerary_reserved > 0)
    {
第六步：        lr_output_message
        ("%d itinerary/itineraries reserved.", itinerary_reserved);
    }
    else
    {
第七步：        lr_error_message("No reservations made.");

        lr_end_transaction("purchase_ticket",LR_FAIL);

第八步：        return 0;
    }
    lr_end_transaction("puchase_ticket",LR_AUTO);
}
…
```

结合上述脚本，人们可以知道。在第一步时，初始化变量并设置值为 0。然后在第二步通过 web_reg_find 保存变量，根据 check point 函数的 savecount 属性保存了匹配的值。那么变量的含义是什么呢？对于 booked_count 值有两种情况：如果其期望值等于实际值，此处即为 booked_count 为 1 或者大于 0，则结果通过；否则，结果失败。也就是说，bookeed_count 的值表示结果的成败。

既然，booked_count 变量值代表了结果的成败，那么可以看到，仅在第三步时，booked_count 的值才会改变。要想取得当前 booked_count 的值，就要利用第四步中的操作，其中 lr_eval_string ("{parameter_name}")函数，可以获取当前值，而 atoi()函数能

将 lr_eval_string()函数获得的字符串值转变为整型。这样就可以将整型化的 booked_count 值赋给最开始定义的整型变量 itinerary_reserved 了。

在第五步,会对 itinerary_reserved 的值进行校验,如果其大于 0,那么说明结果通过,则执行第六步,可借用 lr_output_message()函数向输出日志发送通过信息。否者,执行第七步,借用 lr_error_message()函数向日志发送错误信息。

最后可以用 return(value)函数停止脚本的运行。其中,value 为 0 代表停止当前运行,为-1 代表停止脚本。

上述脚本中的黑体字部分指明了购票事务开始,失败,以及结束的位置。

通过上述方法,就可以利用日志的高级应用功能,自己定义关心的日志内容,这对测试过程无疑是十分有利的。

14.4 高级脚本技术

在 LoadRunner 中高级脚本技术有很多种,本节仅以如何将编写的动态链接库在 LoadRunner 中调用和如何利用 LR 编写 FTP 脚本两个知识点来向读者初步展示其强大的功能。

14.4.1 如何将编写的动态链接库嵌入 LR 中运行

完成动态链接库开发后,动态链接库如何被 LoadRunner 进行调用呢?其实也是很简单的。在 LoadRunner 中的 DLL 调用有局部调用与全局调用,下面介绍局部调用。首先把编译的 DLL 放在脚本路径下面,假设为 MyDll. dll,MyDll. lib. 然后在 Action 中使用 lr_load_dll("MYDll. dll"),此函数可以把 DLL 加载进来,调用 DLL 里面的函数,而 DLL 中的运算是编译级的,所以效率极高。

全局的动态链接库的调用则需要修改 mdrv. dat,路径在 LoadRunner 的安装目录下(LoadRunner/dat directory);在里面修改如下:

```
    [WinSock]
ExtPriorityType=protocol
WINNT_EXT_LIBS=wsrun32.dll
WIN95_EXT_LIBS=wsrun32.dll
LINUX_EXT_LIBS=liblrs.so
SOLARIS_EXT_LIBS=liblrs.so
HPUX_EXT_LIBS=liblrs.sl
AIX_EXT_LIBS=liblrs.so
LibCfgFunc=winsock_exten_conf
UtilityExt=lrun_api
ExtMessageQueue=0
ExtCmdLineOverwrite=-WinInet No
ExtCmdLineConc=-UsingWinInet No
```

```
WINNT_DLLS=user_dll1.dll, user_dll2.dll, …
//最后一行是加载需要的 DLL
```

这样就可以在 LR 中随意调用程序员编写的 API 函数，进行一些复杂的数据加密，准备一些操作，进行复杂的测试。同时如果有大量高复杂的运算也可以放在 DLL 中进行封装，以提高效率。

14.4.2 如何利用 LR 编写 FTP 脚本

以文件上传为例，利用 LR 编写 FTP 脚本的具体步骤如下：

(1) 首先要了解 loadrunner 中的几个 FTP 函数。

在 loadrunner 联机帮助文档的索引中，输入 FTP，此时会看到不下 50 个与 FTP 有关的函数，其实要解决使用 FTP 文件上传这个问题，只需要以下几个步骤就可以了：

① 与 FTP 服务器建立连接。

② 传输文件。

③ 关闭连接。

所以现在只需要关注与这三个步骤有关的函数就可以了。经过挑选终于找到了以下几个函数：

① 与 FTP 服务器建立连接的函数。

```
ftp_logon_ex (FTP * ppftp, char * transaction, char * url, LAST);
```

这个函数主要的功能是建立与 FTP 服务器的连接，其中的参数含义如下：

a. Transaction：为这个连接起一个名字，在这里随便起。

b. url：ftp://username：password@mailserver：port，指定连接用户名、密码、服务器地址、端口。

② 传输文件的函数。

```
ftp_put_ex (FTP * ppftp, char * transaction, char * item_list, LAST);
```

这个函数主要的功能是指定把本地的某个文件上传到服务器的某个目录下。

a. transaction：为这个操作起一个名字。

b. item_list：其中包括

SOURCE_PATH，指定本地上传文件的路径。

TARGET_PATH，指定要上传到服务器的路径。

ENDITEM，标记结尾。

③ 关闭连接的函数。

```
ftp_logout_ex (FTP * ppftp);
```

这个函数的主要功能是断开 FTP 连接。

(2) 模拟文件上传过程。

具体代码如下：

```
Action()
{

unsigned long * ftp_session=NULL;
    ftp_logon_ex(&ftp_session,"ftp_logon","URL=ftp://192.168.0.70:21",LAST);
    ftp_put_ex(&ftp_session, "Ftp_Put",

            "SOURCE_PATH=d:/huruihai.txt",

           "TARGET_PATH=/coreftplite/huruihai.txt", ENDITEM, LAST);

//释放 FTP 连接

    ftp_logout_ex(&ftp_session);

return 0;

}
```

这样,就基本实现了 FTP 脚本的编制。

14.4.3 web_custom_request 使用技巧

初学性能测试时候,第一步必学脚本录制,但一路下来很可能各种录制失败、回放脚本失败的问题层出不穷,究其原因一是 LR 本身存在对测试环境的兼容性问题导致录制失败,更深层次的原因是录制者不清楚 LR 录制脚本的原理,或者不清楚客户端与服务器端之间的请求和应答内容及通信方式,导致一旦出现脚本执行失败便无从下手。例如,进行实际项目的一个接口测试时,假设请求是合作第三方发起的且不容易让第三方提供他们的平台来做测试,这种情况是没办法取录制脚本的,只能选择手动编写脚本实现。得知接口是使用 HTTP 的 post 方法,想到了 web_submit_form()和 web_submit_data()两个函数,它们实现了 HTTP 请求中的 post 方法,现在它们都提交表单到某个页面,但现在被测试的仅是个 HTTP 接口,显然这两个函数都无法满足需要。

这种情况下就可以用 web_custom_request(),这个函数的作用是自定义 HTTP 请求规则,甚至可以说 web_custom_request()函数是一个可以用于自定义 HTTP 请求的“万能”函数,具有 web_link()、web_url()、web_submit_data()函数的功能,可以自由定义需要的 HTTP 的 get 和 post 请求。

下面即为 web_custom_request 函数语法详解。

语法如下:

```
Int web_custom_request (const char * RequestName, <List of Attributes>,
[EXTRARES, <List of Resource Attributes>,] LAST);
```

返回值如下：

LR_PASS(0)代表成功。

LR_FAIL(1)代表失败。

参数如下：

(1) RequestName：步骤的名称，VuGen 中树状视图中显示的名称。

(2) List of Attribute：支持的属性有以下几种。

① URL。

页面地址。

② Method。

页面的提交方式，post 或 get。

③ EncType。

编码类型。此参数给出一个内容类型(Content-Type)，指定其作为回放脚本时 Content-Type 请求头的值，例如 text/html。web_custom_request 函数不处理未编码的请求体。请求体参数将会使用已经指定的编码方式。因此，如果指定了不匹配 HTTP 请求体的 EncType，会引发服务端的错误。通常建议不要手动修改录制时的 EncType 值。任何对于 EncType 的指定都会覆盖 web_add_[auto_]header 函数指定的 Content-Type。当指定了 EncType=(空值)时，不会产生 Content-Type 请求头。当省略了 EncType 时，任何一个 web_add_[auto_]header 函数都会起作用。如果既没有指定 EncType 也没有 web_add_[auto_]header 函数且 Method=POST，application/x-www-form-urlencoded 会作为默认值来使用。其他情况下，不会产生 Content-Type 请求头。

④ BodyFilePath。

作为请求体传送的文件路径。它不能与下面的属性一起使用：Body 或者其他 Body 属性或 Raw Body 属性，包括 BodyBinary，BodyUnicode，RAW_BODY_START 或 Binary =1。

⑤ UserAgent。

用户代理，它是一个 HTTP 头的名字，用来标识应用程序，通常是浏览器，它呈现的是用户和服务器的交互。例如：头信息 User-Agent：Mozilla/4.0 (compatible；MSIE 6.0；Windows NT 5.0)识别的是 Windows NT 下的 IE 浏览器 6.0。其他 User-Agent 的值用来描述其他浏览器，或者非浏览器程序。通常，一个应用程序中所有的请求都使用相同的用户代理，录制者作为一个运行时的参数来指定(Run-Time Setting—Browser Emulation—User Agent)。不管怎么说，即使是在一个简单的浏览器进程中，仍有可能会用到直接与服务器交互的非浏览器组件(例如 ActiveX 控件)，通常它们有着不同于浏览器的用户代理属性。指定 UserAgent 表示这是一个非浏览器的请求。指定的字符串被 HTTP 头 User-Agent：使用，在某些情况下，它同时会影响回放脚本时的行为。例如，不使用浏览器缓存，假设指定的 URL 属于资源等。(LoadRunner 本身不检查指定的字符串与浏览器本身的值是否相同。)

⑥ Binary。

Binary=1 表示页面请求体中的每一个以 file://x/##形式出现的值(在这里##

代表两个十六进制数字),都会被替换为单字节的十六进制值。如果 Binary=0(默认值),所有的字符序列只是按照字面的值传递的。需要注意双斜杠的用法。在 C 编译器中双斜杠被解释为单斜杠。如果不需要零字节,单斜杠可以在 Binary 不等于 1 的情况下使用(例如,使用\x20 代替 file://x20/)。如果需要零字节,那么只能使用 file://x00/且设置 Binary=1,\x00 在逻辑上会被截断。

⑦ ContentEncoding。

指定请求体的使用指定的方式(gzip 或者 deflate)进行编码(例如,压缩),相应的 Content-Encoding:HTTP 头会和此请求一起发送。这个参数适用于 web_custom_request 和 web_submit_data。

⑧ FtpAscii。

1 使用 ASCII 模式处理 FTP 操作;0 使用二进制模式。

⑨ TargetFrame。

当前链接或资源所在 Frame 的名称。除了 Frame 的名字,还可以指定下面的参数。

_BLANK:打开一个空窗口。

_PARENT:把最新更改过的 Frame 替换为它的上级。

_SELF:替换最新更改过的 Frame。

_TOP:替换整个页面。

⑩ RecContentType。

录制脚本时响应头的内容类型。例如 text/html、application/x-javascript 等。当没有设置 Resource 属性时,用它来确定目标 URL 是否是可记录的资源。此属性包含主要的和次要的资源。最频繁使用的类型是 text、application、image。次要的类型根据资源不同变化很多。例如:RecContentType=text/html 表示 html 文本。RecContentType=application/msword 表示当前使用的是 Msword。

⑪ Referer。

当前页面关联的页面。如果已经显式指定了 URL 的地址,此项可以省略。

⑫ Resource。

指示 URL 是否属于资源。1 是;0 不是。设置了这个参数后,RecContentType 参数被忽略。Resource=1,意味着当前操作与所在脚本的成功与否关系不大。在下载资源时如果发生错误,是当作警告而不是错误来处理的;URL 是否被下载受 Run-Time Setting—Browser Emulation—Download non-HTML resources 这个选项的影响。此操作的响应信息是不作为 HTML 来解析的。Resource=0,表明此 URL 是重要的,不受发送请求(RTS)的影响,在需要时也会解析它。

⑬ ResourceByteLimit。

Web 页面下载资源的极限大小。当达到设置的极限后,无法下载其他资源,仅仅对需要下载的资源有效。下载过程:如果总计下载大小小于极限值,则正常开始下载。如果下载时达到了设置的极限值,资源大小可知(在 HTTP 响应头中指定了 Content-Length),在这种情况下,如果只需要一个缓冲区,那么下载可以正常完成。如果需要的不止一个缓冲区,或者资源大小不可知,下载就会中断同时关闭当前连接。这个特性可以

用来模拟用户不等待一个页面下载完成时导航到另一个页面的情况。在 HTTP 模式中无法使用，在 Concurrent Groups（Vuser 脚本中的一个区，此区中的所有函数并发执行）区中也无法使用。仅仅适用于 Sockets 的回放，WinInet 也是不适用的。

⑭ Snapshot。

快照的文件名。

⑮ Mode。

两种录制级别 HTML、HTTP。HTML 级别——在当前 Web 界面上录制直观的 HTML 动作。以一步步的 web_url、web_link、web_image、web_submit_form 来录制这些动作。VuGen 仅仅录制返回 HTML 页面的请求，不处理脚本和应用程序。HTTP 级别——VuGen 把所有的请求录制为 web_url 指令，不生成 web_link、web_image、web_submit_form 这些函数。这种方法更为灵活，但是生成的脚本不够直观。

⑯ ExtraResBaseDir。

根 URL 放在 EXTRARES 组里。它是用来解析相对 URL 的（译者加：类似于 Windows 的相对路径和绝对路径）。URL 可以是绝对路径（例如 http://weather. abc. com/weather/forecast. jsp? locCode＝LFPO），也可以是相对路径（例如 forecast. jsp? locCode＝LFPO）。真正的 URL 的下载是通过绝对路径进行的，所以相对 URL 路径必须使用根路径 URL 去解析。例如，使用 http://weather. abc. com/weather/作为根路径来解析 forecast. jsp? locCode＝LFPO，最后的 URL 是：http://weather. abc. com/weather/forecast. jsp? locCode＝LFPO。如果没有指定 ExtraResBaseDir，默认的根 URL 是主页面的 URL。

⑰ Body。

请求体。不同的应用中，请求体分别通过 Body、BodyBinary 或者 BodyUnicode 参数来传递。请求体可以只使用其中一个参数，也可以使用一连串分开的参数组成多请求体。例如：

```
web_custom_request(
    ……
    "BodyUnicode=REPRICE"
    "BodyBinary=\\x08\\x00\\xCC\\x02\\x00\\x00"
    "Body=.\r\n"
    "-dxjjtbw/(.tp?eg:ch/6--\r\n",
LAST);
```

在上面的代码中，使用了三个参数来划分请求体，一个是 Unicode 段，一个是二进制段，最后一个是常规的字符串。最终的请求体是这三个参数按照在函数中的顺序连接起来的值。还有一个很少用到的参数，Binary。它也能描述二进制请求体，但只允许函数中只有一个请求体参数。所有的请求体都是 ASCII 字符，以 null 结束。

Body 表示规则的、可打印的字符串，无法表示空字节，所有的字符都以一个反斜杠表示。注意：在旧的脚本中，可以看见不可打印的字符在请求体中以十六进制方式进行编码。例如\\x5c，在这种情况下，必须使用 Binary＝1 来标识。空字节使用 file://0. 0. 0.

0/来表示。相反，新脚本则会把请求体分开放在不同的参数中（"Body=…"，"BodyBinary=…"，"Body=…"）。

BodyBinary 表示二进制代码。不可打印的字符在请求体中以十六进制方式 file://xhh/进行编码。在这里 HH 表示十六进制值。空字节使用 file://0.0.0.0/来表示。

BodyUnicode——美国英语，特指拉丁 UTF-16LE(little-endian)编码。这种编码方式会在每个字符末尾附加一个 0 字节，以便使字符更可读。但是在 VuGen 中实际的参数是把所有的 0 字节都去掉的。但是在发送给 Web 服务器之前，web_custom_request 函数会重新添加 0 字节。对于不可打印的字符，使用单反斜杠表示，无法表示空字节。

注意：如果请求体大于 100K 字节，会使用一个变量来代替 Body 参数。变量是在 lrw_custom_body.h 中定义的。

⑱ Raw Body。

请求体是作为指针传递的，此指针指向一串数据。二进制的请求体可以使用 BodyBinary 属性来发送（或者使用 Body 属性来传递，前提是必设置 Binary=1）。无论如何，这种方法需要使用转义字符反斜杠把不可打印的字符转换为 ASCII 字符。为了能有一种更简便的表现原始数据的方式，Raw Body 属性应运而生，可以传递指向二进制数据的指针。使用 4 个连续的参数集来表示指针，而且必须按照顺序排列：

```
RAW_BODY_START
指向数据缓冲区的指针
(int) 长度
RAW_BODY_END
```

例子：

```
char * abc= .../* a pointer to the raw data * /
web_custom_request("StepName",
"URL= http://some.url ",
"Method= POST",
RAW_BODY_START,
"abc",
3,
RAW_BODY_END,
LAST);
```

在应用中，即使设置了数据的长度为 0，指针也必须有值，不能为空。在 Binary=1 时，不能使用上面的语法传递原始数据。数据缓冲区中的数据不能使用参数化。也就是说，缓冲区中的任何参数（例如{MyParam}）不能被正确地替代为相应的值，只会以字面值发送。

(3) EXTRARES：表明下面的参数将会是 List Of Resource Attributes 了。

(4) LAST：结尾的标识符。

(5) List of Resource Attributes：

仅仅当 Recording Options—Recording—HTML-based script—Record within the

current script step 选项被选中时，List of Resource Attributes 才会被插入代码中。Web 页面中的非 HTML 机制产生了资源列表，包含了 Javascript，ActiveX，Java applets and Flash 所请求的资源。VuGen's 的 Recording 选项中，可以设置把这些资源录制在当前的操作中（默认是此设置）还是作为单独的步骤来录制。

14.4.4 特殊的录制脚本方法

本节将以 LoadRunner 对 Foxmail 程序进行录制为例，介绍一种特殊的录制脚本的方法。

运行 LoadRunner，选择录制脚本。在录制协议选择对话框中，依然选择 POP3，如图 14-12 所示。

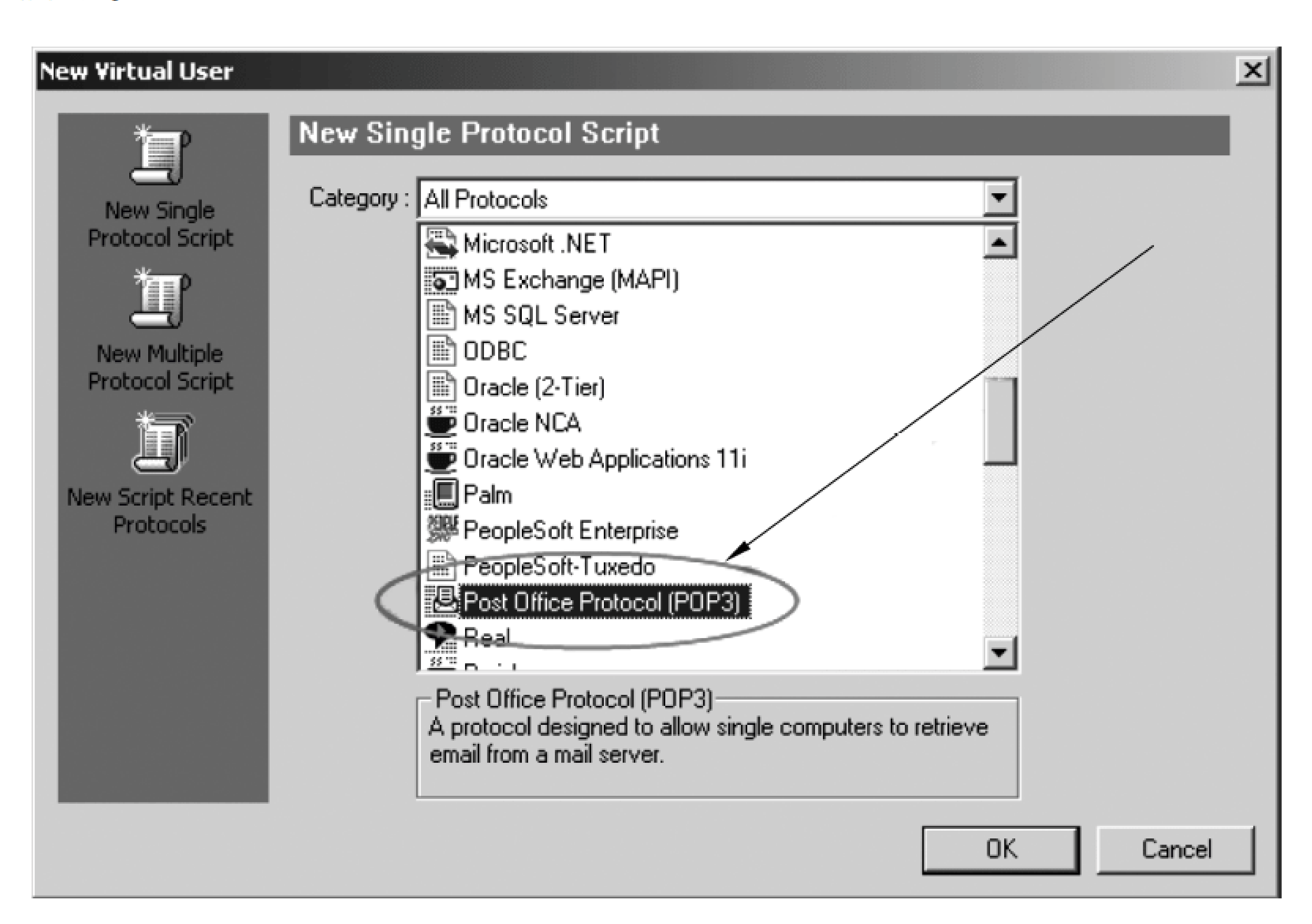


图 14-12 选择录制协议

特殊之处在于下一步。接着在选择录制的程序时，不再按典型的方法选择 Foxmail .exe，而是选择了 $LOADRUNNER\bin\wplus_init_wock.exe。其中 $LOADRUNNER 是 LoadRunner 的安装目录。通常 $LOADRUNNER 表示 C:\Program Files\Mercury Interactive\Mercury Load Runner，相关设置如图 14-13 所示。

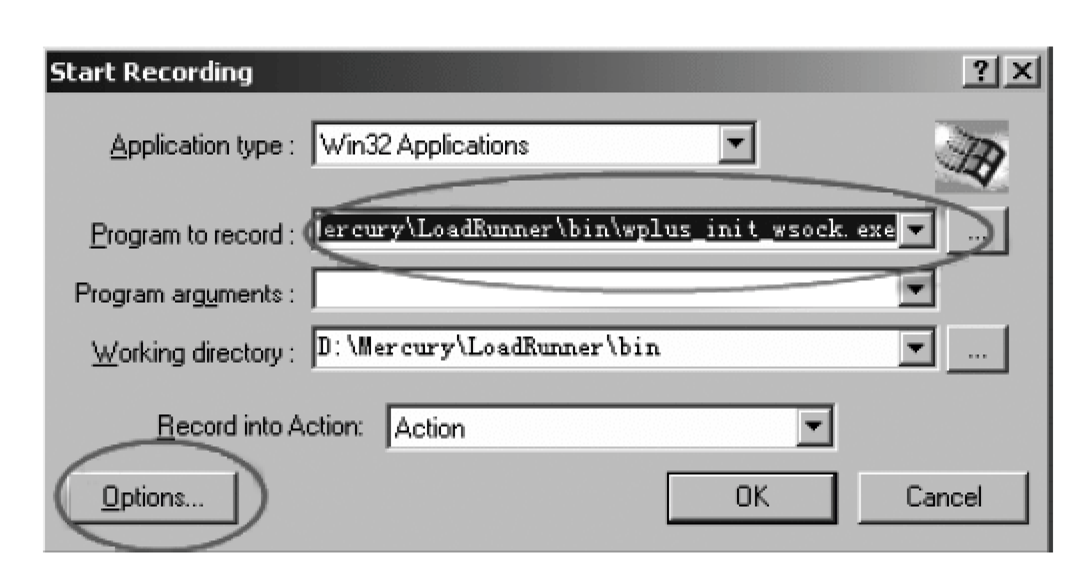


图 14-13 选择录制的程序

紧接着，单击 Options 按钮，进入录制选项设置对话框。在该对话框中，选择左边的一个选项 Port Mapping，然后单击 New Entry，如图 14-14 所示。

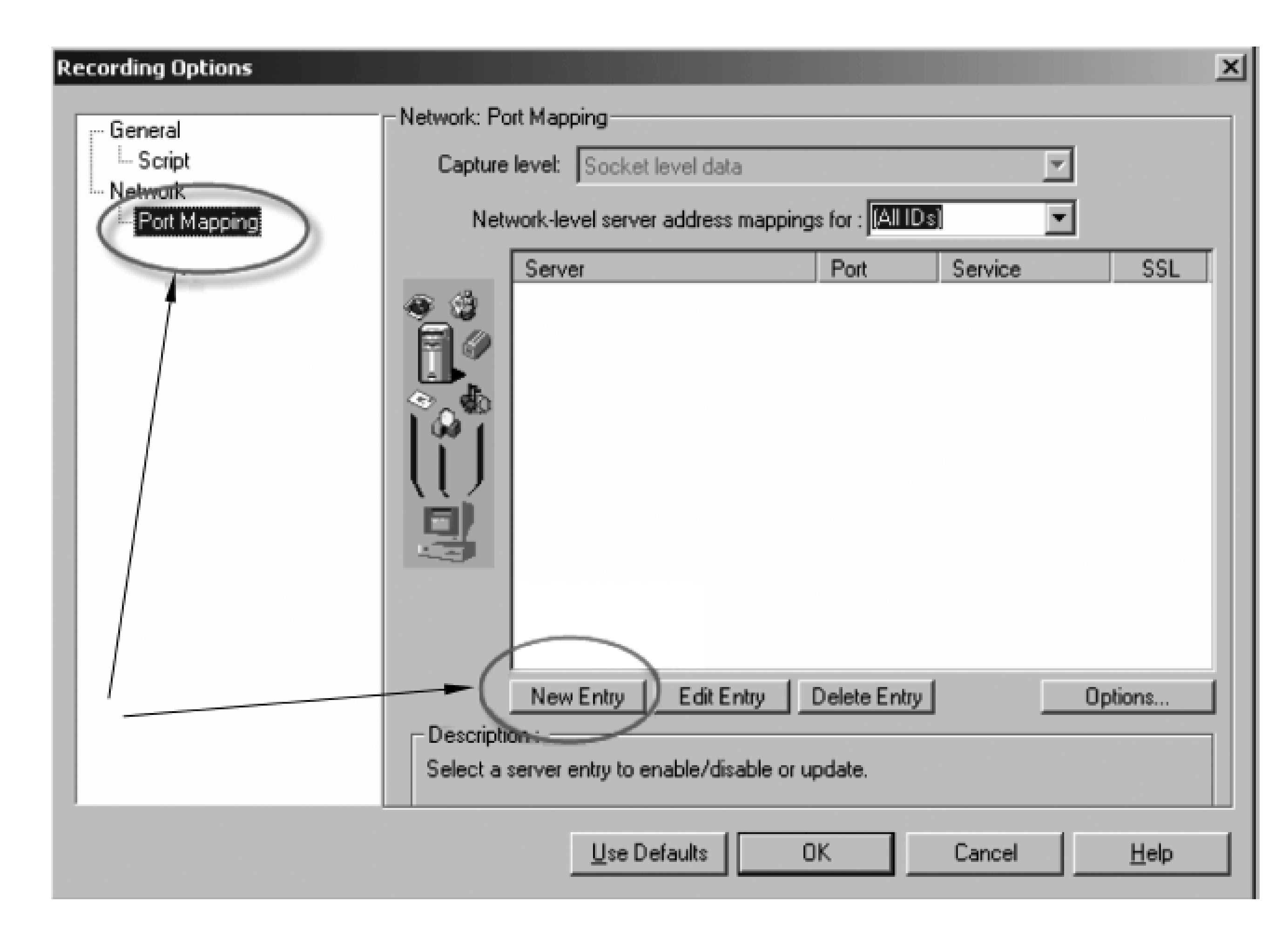


图 14-14　录制选项设置

在接着出现的对话框中，进行以下设置，如图 14-15 所示。

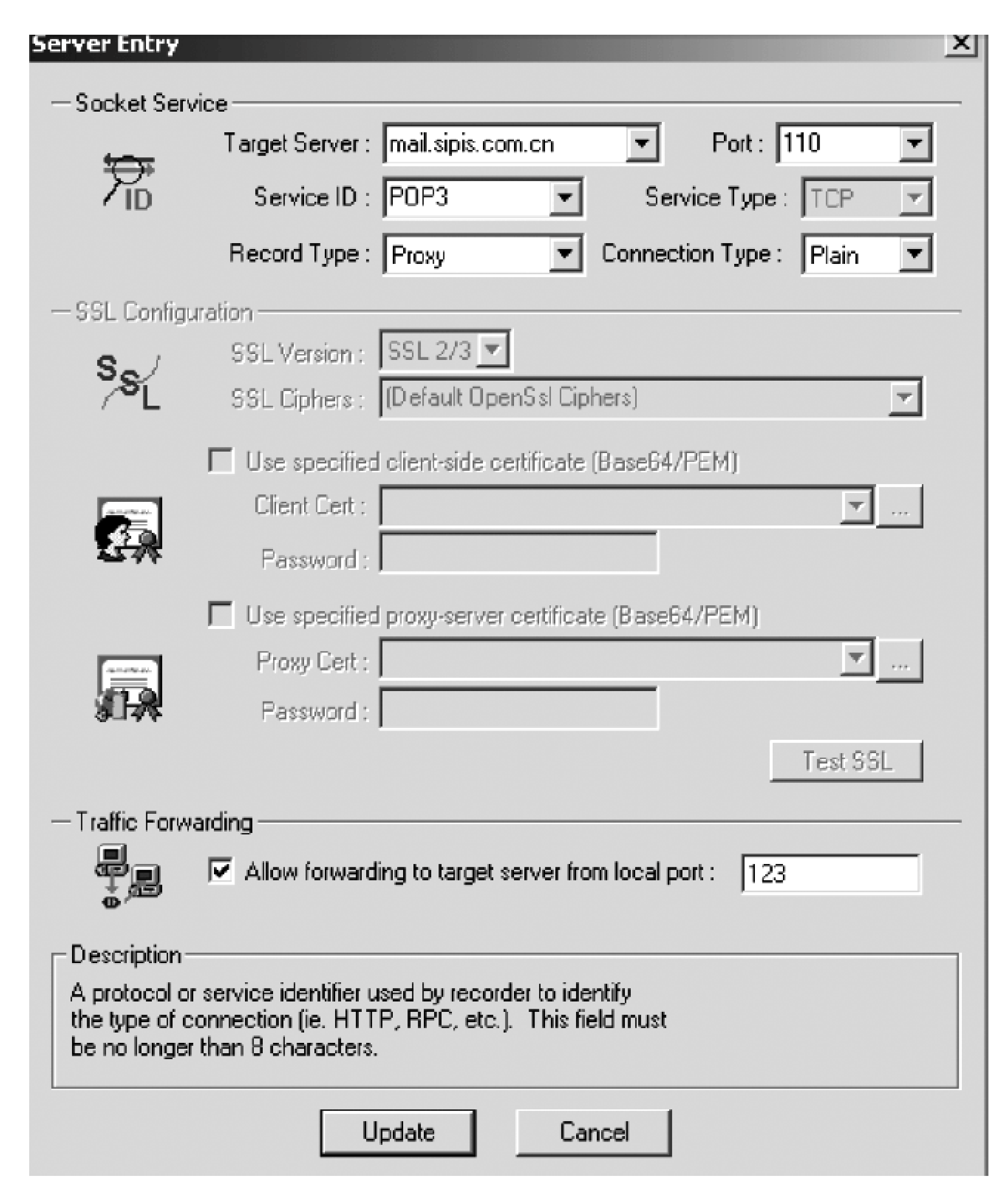


图 14-15　录制选项设置

对设置进行总结如下：

Target Server，选择了邮件服务器的地址，这里可以输入 IP 地址。由于 POP3 的端口号是 110，在 Port 里输入 110。

ServiceID 选择了 POP3。

在底部的 Traffic Forwarding 中，把 Allow forwarding to target server from local port 选中，并设置了一个端口。这个端口可以随便设置，如设置成了 123。

其意义在于通过这样的设置告诉 LoadRunner：把所有转发到 LoadRunner 所在的机器的 123 端口的请求，都转发到 mail.sipis.com.cn 的 110 端口，其原理如图 14-16 所示。

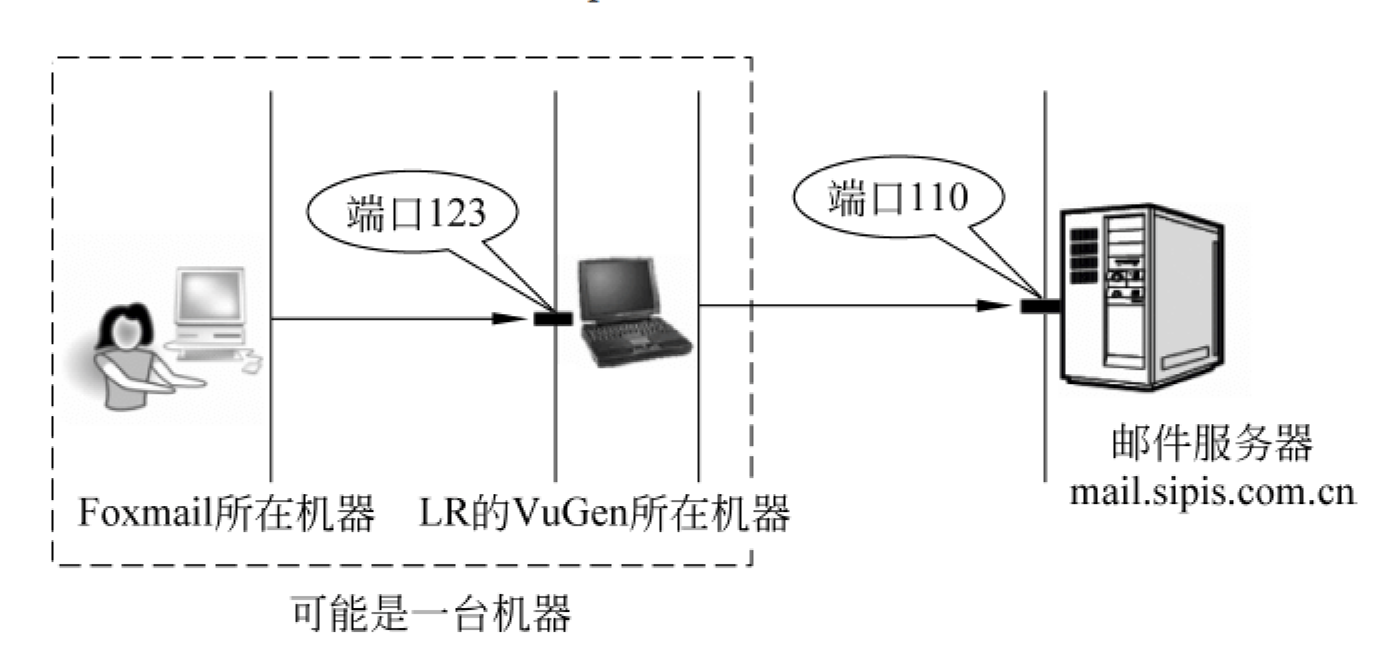


图 14-16 设置的原理示意图

做完 LoadRunner 的设置后，需要把 Foxmail 的一些设置进行更改。选择 Foxmail 的“账户”→“属性”，如图 14-17 所示。

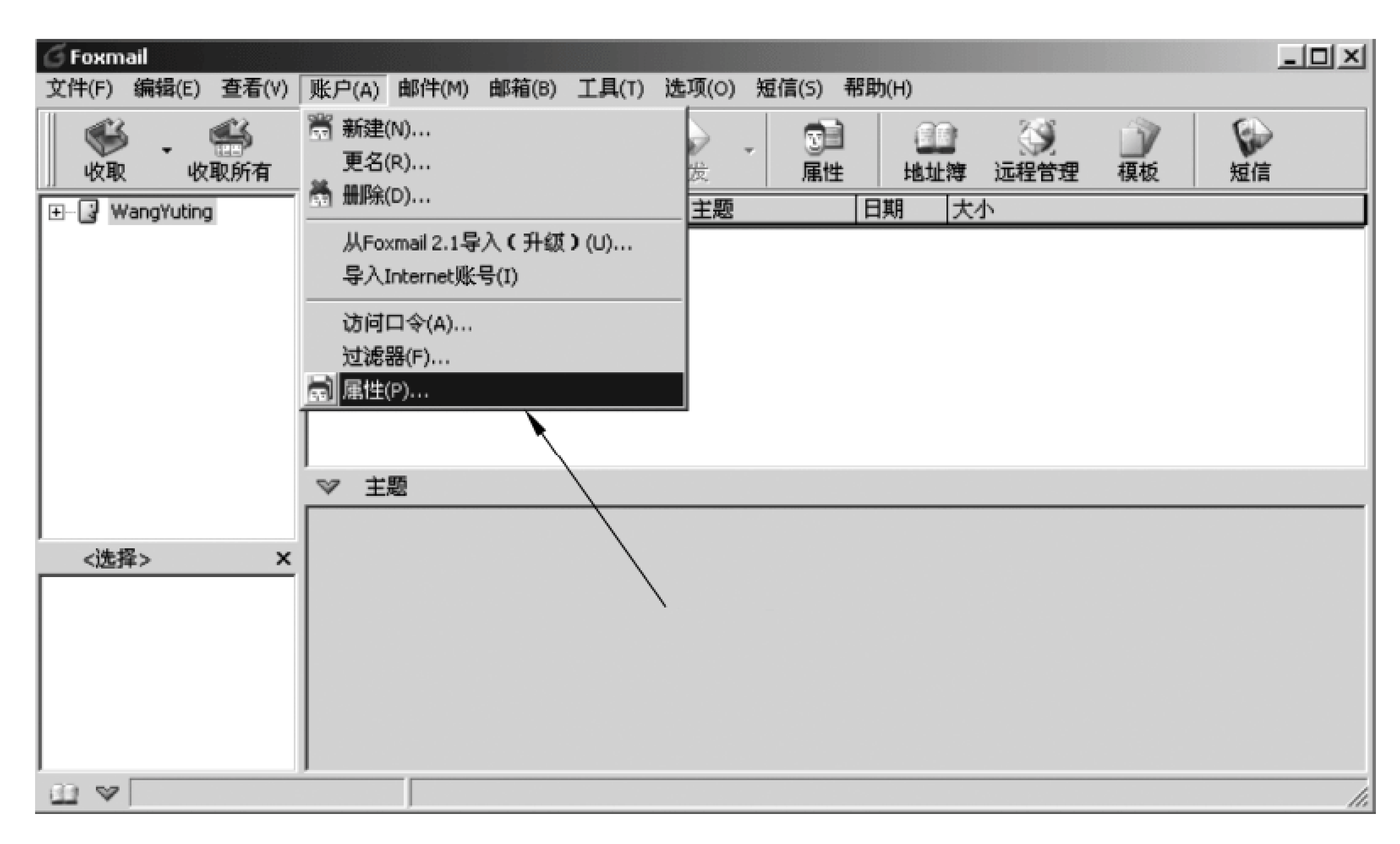


图 14-17 更改 Foxmail 的设置

在邮件服务器设置上，把接收邮件服务器设置成 localhost 或者 127.0.0.1。然后单击“高级”按钮，如图 14-18 所示。

在高级设置中，把 POP3 的服务器端口设置成 123，如图 14-19 所示。这个 123 要和在 LoadRunner 录制选项里面的 Allow forwarding to target server from local port 中的

端口设置一致。

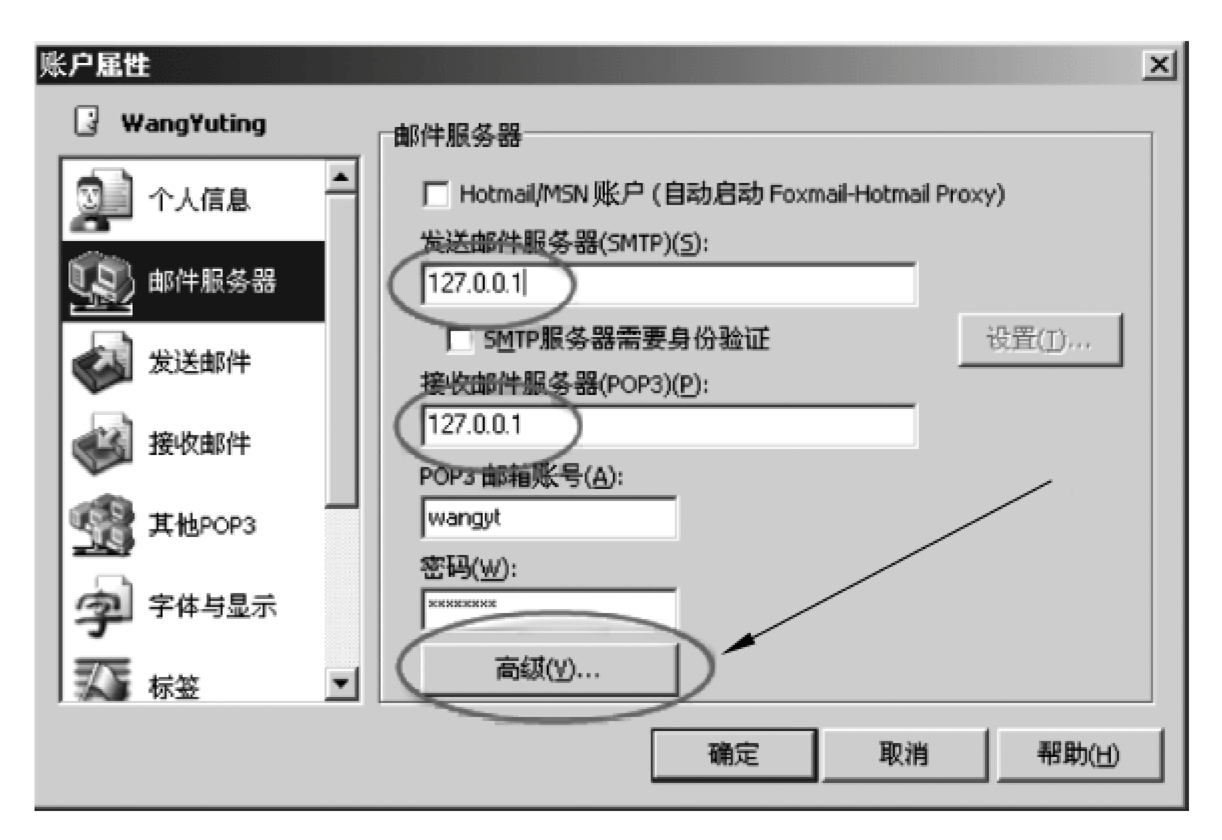


图 14-18　设置接收邮件服务器

准备工作结束，接着启动 LoadRunner 的录制功能。LoadRunner 激活 $LOADRUNNER\bin\wplus_init_wsock.exe，如图 14-20 所示。它实际上是一个代理服务器 Proxy 程序。

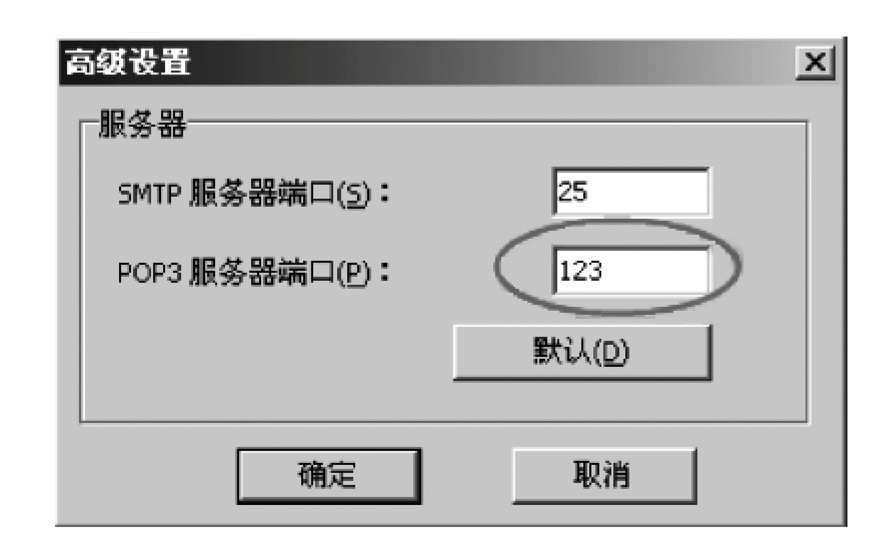


图 14-19　设置 POP3 服务端口号

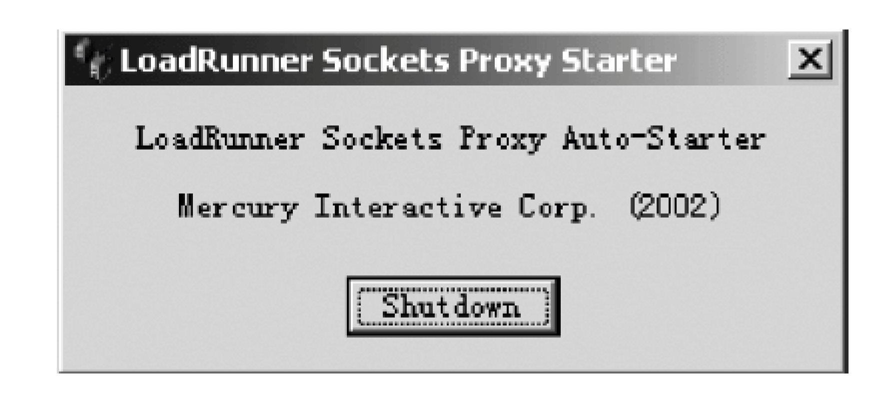


图 14-20　激活 wplus_init_wsock.exe

LoadRunner 会做一些初始化的工作。结果导致什么动作都没有做的时候，LoadRuner 已经捕获了 14 个网络通信包了，如图 14-21 所示。

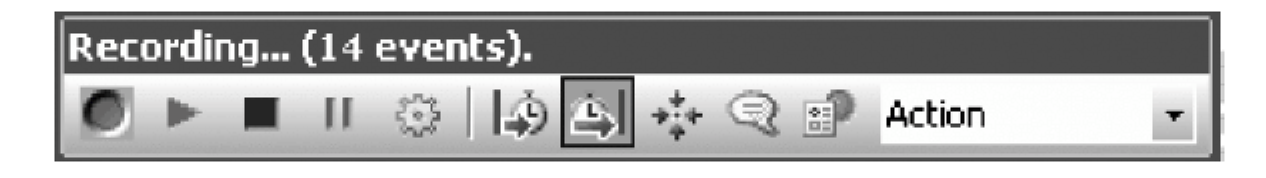


图 14-21　LoadRunner 对初始化产生的通信包的捕获结果

单击 Foxmail 中的“收取”按钮，如图 14-22 所示。根据设置，Foxmail 会向 127.0.0.1 即本机的 123 端口发送 POP3 的取信命令。这些命令都被 wplus_init_wsock.exe 捕获了。然后 wplus_init_wsock.exe 把这些命令转发到 mail.sipis.com.cn 的 110 端口。那里是真正的邮件服务器。邮件服务器把信取出后，发给 wplus_init_wsock.exe，然后 wplus_init_wsock.exe 把信转发给 Foxmail，其结果就是 Foxmail 正确地收取到了邮件。

LoadRunner 显示捕获了 33 个网络包，如图 14-23 所示。扣除上面 14 个，实际上捕获了 19 个网络包。这些网络包就是 Foxmail 发出的取信请求和邮件服务器传递给 Foxmail 的邮件，只不过这些请求都经过 wplus_init_wsock.exe 这个 proxy 程序进行转



图 14-22 选择收取

发。wplus_init_wsock.exe 在转发的同时，悄悄把这些通信包记录下来，并转化成了脚本。这就是 LoadRunner 录制脚本的真正原理所在。



图 14-23 LoadRunner 捕获的网络包数量

然后，停止录制看以下最后的结果，如图 14-24 所示。

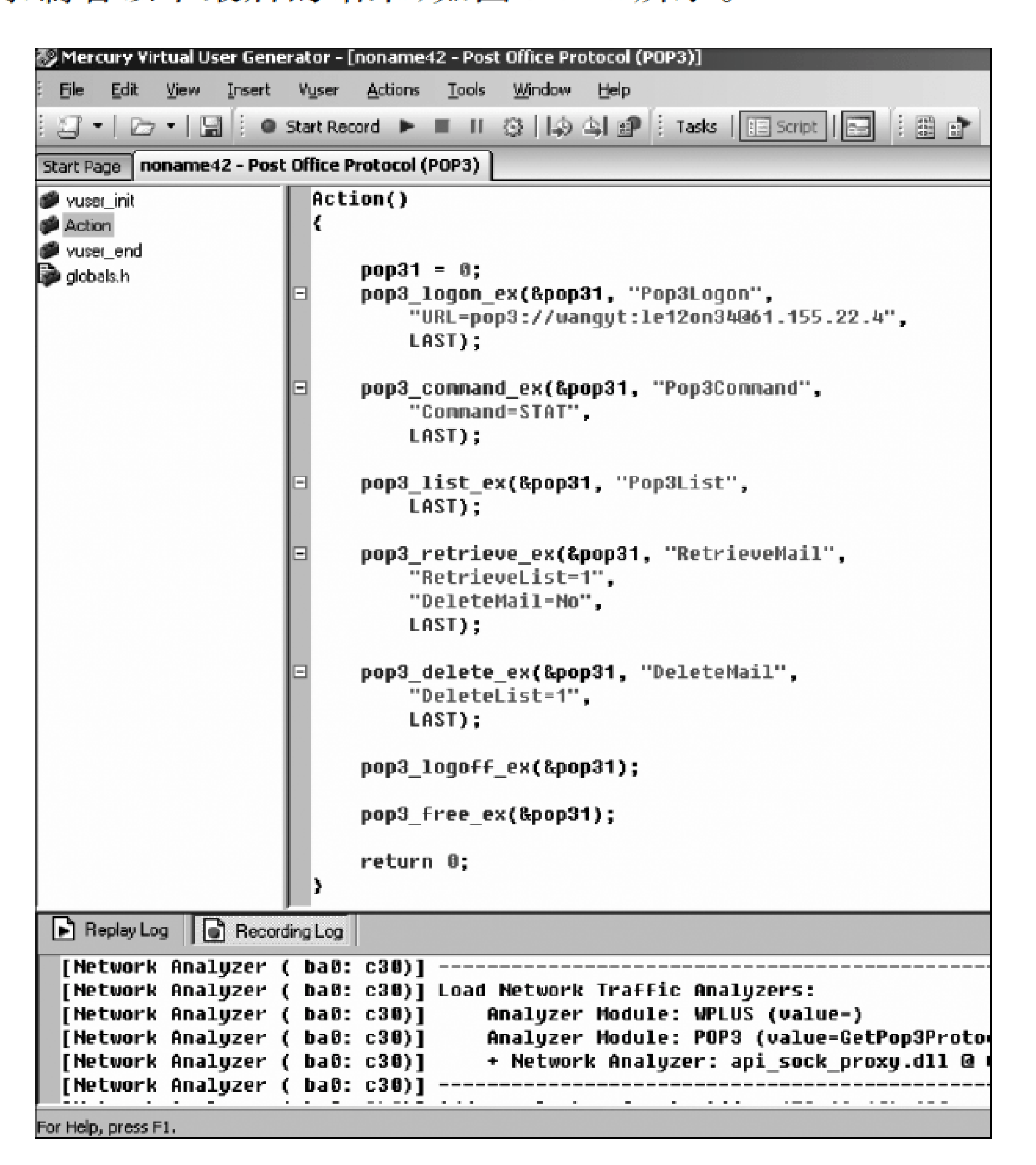


图 14-24 最终的录制结果

最后,需要关闭 wplus_init_wsock.exe。否则再次录制脚本的时候,由于端口已经被占用,导致录制无法进行,但是 LoadRunner 不会报任何错误,这是最容易犯的错误。

14.5 本章小结

在整个性能测试的过程中,测试脚本的编写扮演着十分重要的角色。因为它是实现预定测试方案的具体步骤。其成败直接关系到,测试方案能不能被忠实地执行。

因此,本章从参数化脚本,手工关联和自动关联,日志高级应用,高级脚本技术几个方面,对测试脚本编写这一内容进行了介绍。对于参数化脚本,说明了参数化的目的,什么时候进行参数化以及怎样进行参数化;对于关联,说明了其原理,目的,以及手工关联和自动关联的具体操作步骤。另外本章还介绍了 loadRunner 中日志的高级应用和高级脚本技术的应用。

第 15 章　测试场景设计与执行

15.1　场景设计介绍

15.1.1　新建场景

当虚拟用户脚本开发完成后，使用 LoadRunner 的 Controller 组件将这个执行脚本的用户从单人转化为众人，从而模拟大量用户操作，进而形成负载。需要对这个负载模拟的方式和特征进行配置，从而形成场景。场景(Scenario)是一种用来模拟大量用户操作的技术手段，通过配置和执行场景向服务器产生负载，验证系统各项性能指标是否达到用户要求，而 Controller 组件可以帮助人们对场景的设计、执行及监控进行管理。

使用 Controller 组件管理场景主要分为场景设计和场景监控两部分，最后通过运行场景完成性能测试的执行。

Controller 控制器提供了手动和面向目标两种测试场景。一般情况下使用手动场景设计方法来设计场景，手动设计场景最大的优点是能够更灵活地按照需求来设计场景模型，使场景能更好地接近用户的真实使用。面向目标场景则是测试性能能否达到预期的目标，在能力规划和能力验证的测试过程中经常使用到。

创建场景有两种方式：

(1) 通过 VuGen 直接转换当前脚本进入场景。

在 Tools 菜单下打开 Create Controller Scenario，就可以将当前脚本转化为场景，如图 15-1 所示。

接着需要设置场景的类型、负载服务器的地址、脚本组的名称以及结果的保存地址。如果选择 Manual Scenario(手工场景)，那么还需要进一步设置手工场景模拟的用户数，如图 15-2 所示。

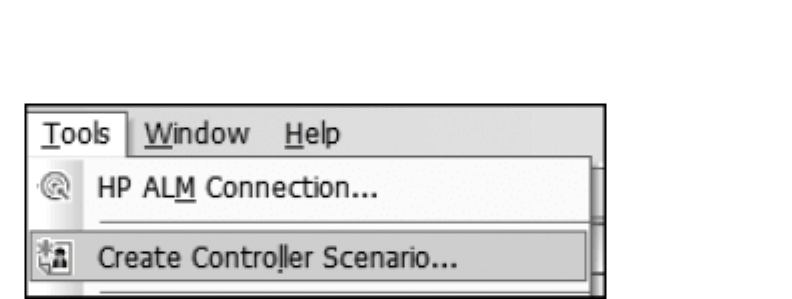


图 15-1　在 VuGen 中直接生成场景

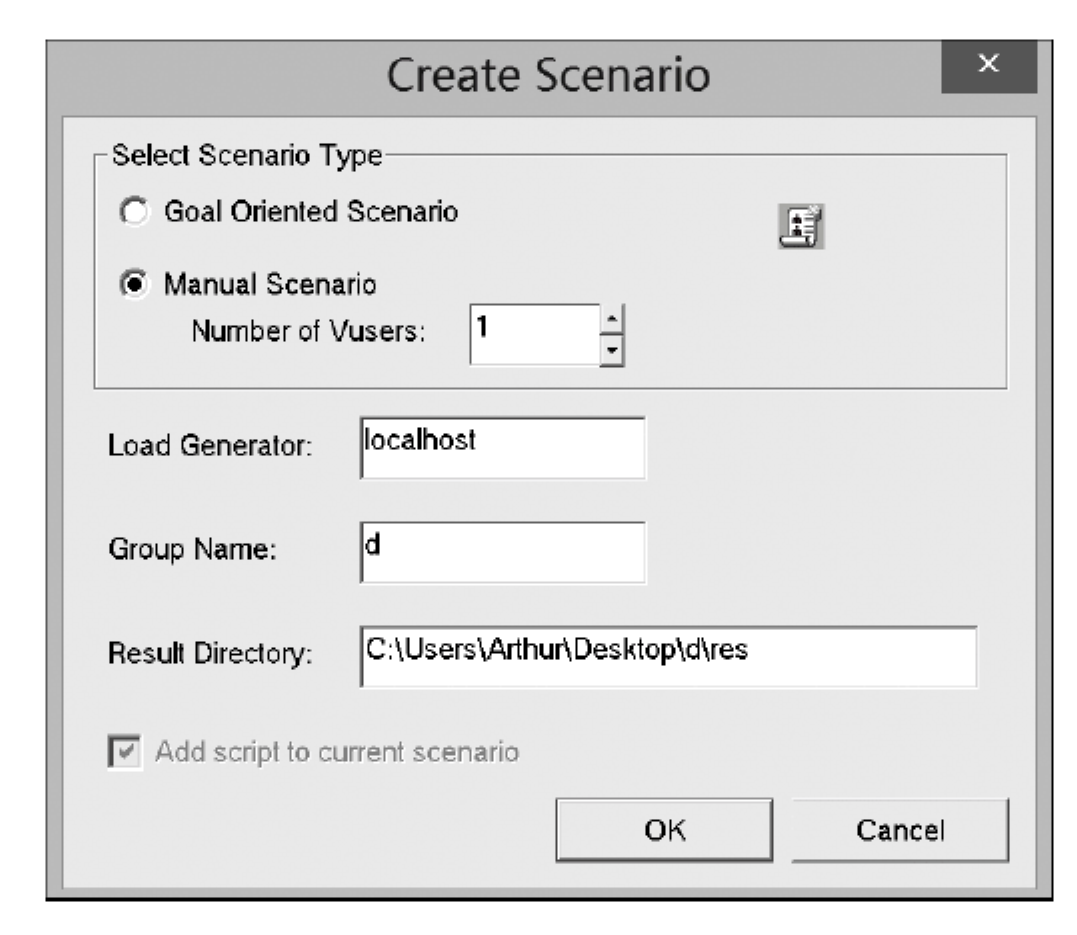


图 15-2　VuGen 中设置手工场景属性

(2) 打开 Controller 新建场景,在弹出的新场景设置对话框中可以设置场景类型和对应的脚本,如图 15-3 所示。

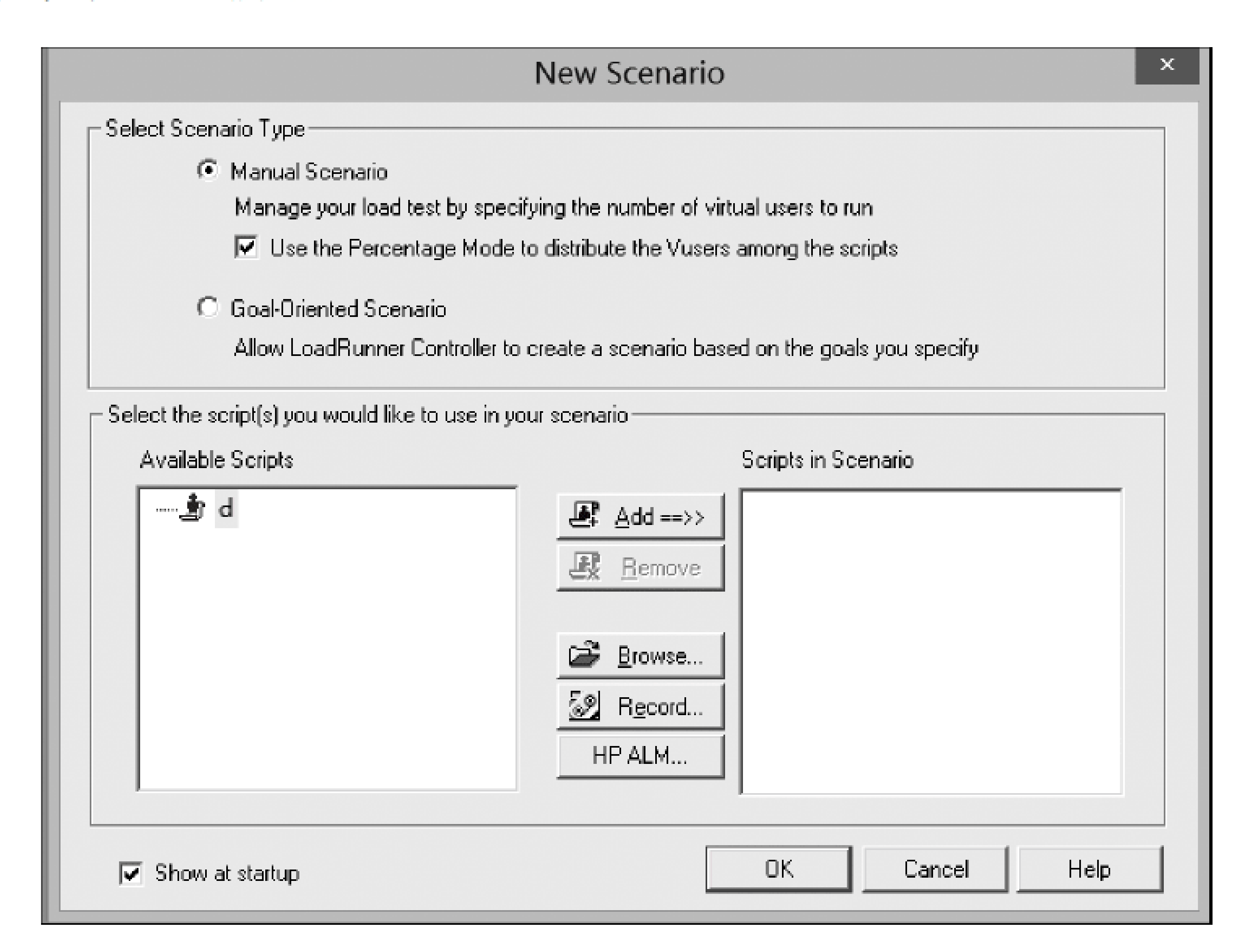


图 15-3 Controller 中的新建场景设置

将左侧 Available Scripts 脚本列表中的对应脚本添加到右侧的 Scripts in Scenario 场景列表中即可,也可以通过单击 Browse 按钮打开已经存在的脚本或者调用 Quality Center 上的脚本。

1. 目标场景

所谓目标场景(Goal Scenario),就是设置一个运行目标,通过 Controller 的 Auto Load 功能进行自动化负载,如果测试的结果达到目标,则说明系统的性能符合测试目标,否则就提示无法达到目标。

目标场景是定性型的性能测试,人们只关心最后性能测试的结论是否符合性能需求,常常用在验收测试的场合。

在新建场景时选择目标场景,并添加需要执行负载的脚本。确定后进入目标场景设置窗口,如图 15-4 所示。

在目标场景中主要设置一个需要测试的目标,Controller 会自动逐渐增加负载,测试系统能否稳定地达到预先设定的目标。

单击 Edit Scenario Goal 按钮打开目标场景编辑对话框,如图 15-5 所示。

在目标场景中最重要的就是目标类型,目标场景提供了 5 种目标,如图 15-6 所示,每种目标都有自己独立的设置。

1) Virtual Users

该参数表示虚拟用户数,被测系统所需要支持的用户数。这里只需要填写系统能够达到的用户数目即可。

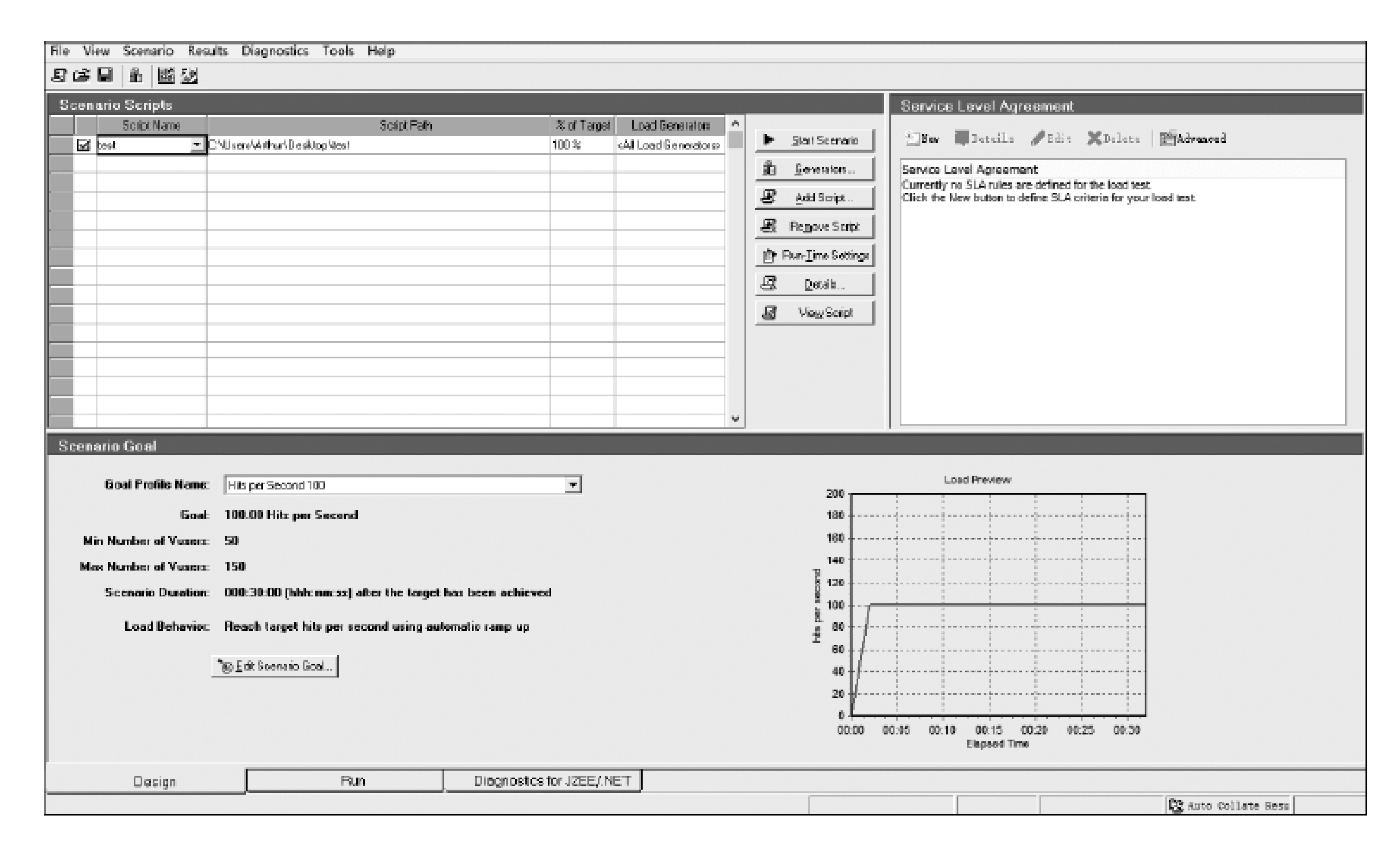


图 15-4　目标场景设置窗口

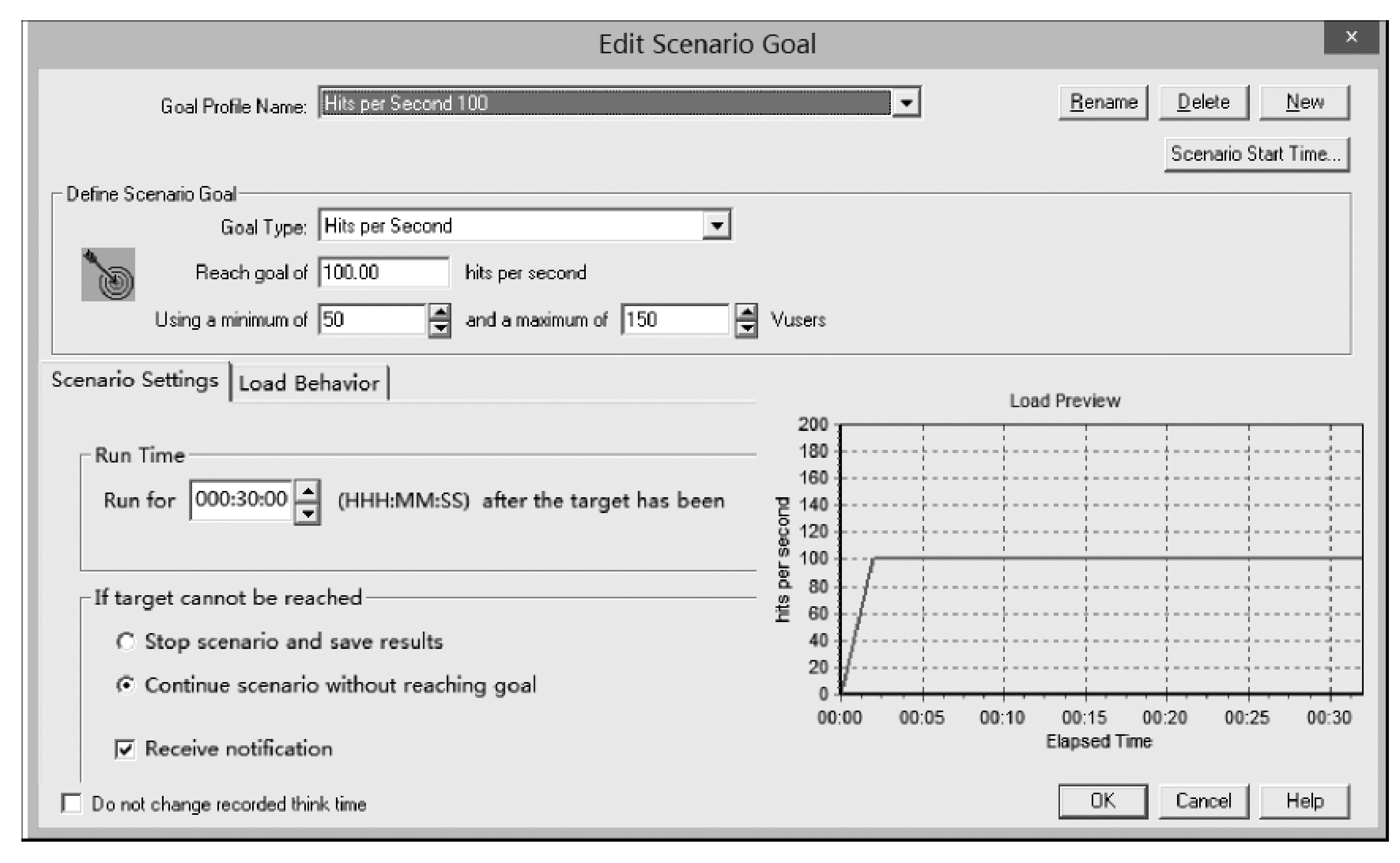


图 15-5　设置目标场景中的目标

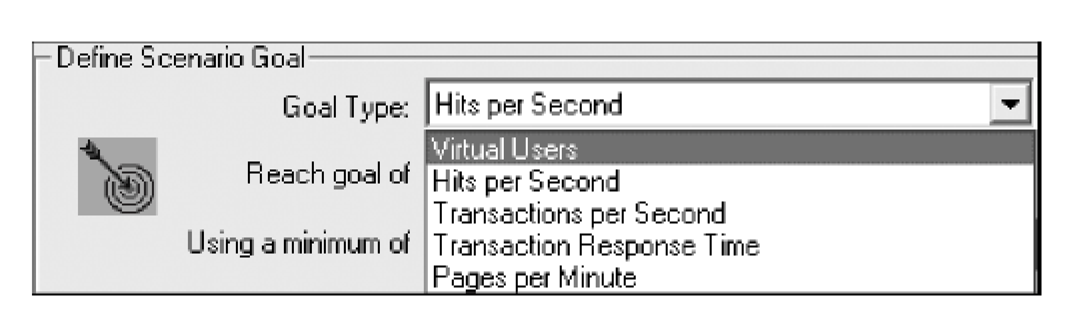


图 15-6　目标场景中提供的目标类型

2）Hits per Second

该参数表示每秒单击数，是指在一秒钟能够做到的单击请求数目，即客户端产生的每秒请求数(正常情况下每秒单击数等同于服务器请求响应数)。除了要设置单击的指标，

还需要设置在线用户的上下限，场景运行时会自动调整用户数，来测试在一定的用户范围内系统是否都能达到定义的目标。

3）Transactions per Second

该参数每秒事务数，一个事务代表完成一个操作，每秒事务数反映了系统的处理能力。当脚本中含有事务函数时才可以使用，这里需要事务名称、TPS 指标及需要完成该指标的用户数。

4）Transaction Response Time

该参数表示事务的响应时间，反映了该系统的处理速度以及执行一个操作所需要花费的时间。和 Transactions per Second 类似，当脚本中含有事务函数时，可以设定事务响应时间的指标。

5）Pages per Minute

该参数表示每分钟页面的刷新次数，反映了系统在每分钟下所能提供的 Page（页面）处理能力。页面的生成能力反映了一个系统的整体处理能力，一个页面请求包含了多个单击请求。

当设置完成性能的目标后，还需要设置一下场景运行时的模式，这里分为两大部分。

（1）Scenario Settings（场景设置）提供对目标场景运行的一些基础设置，如图 15-7 所示。

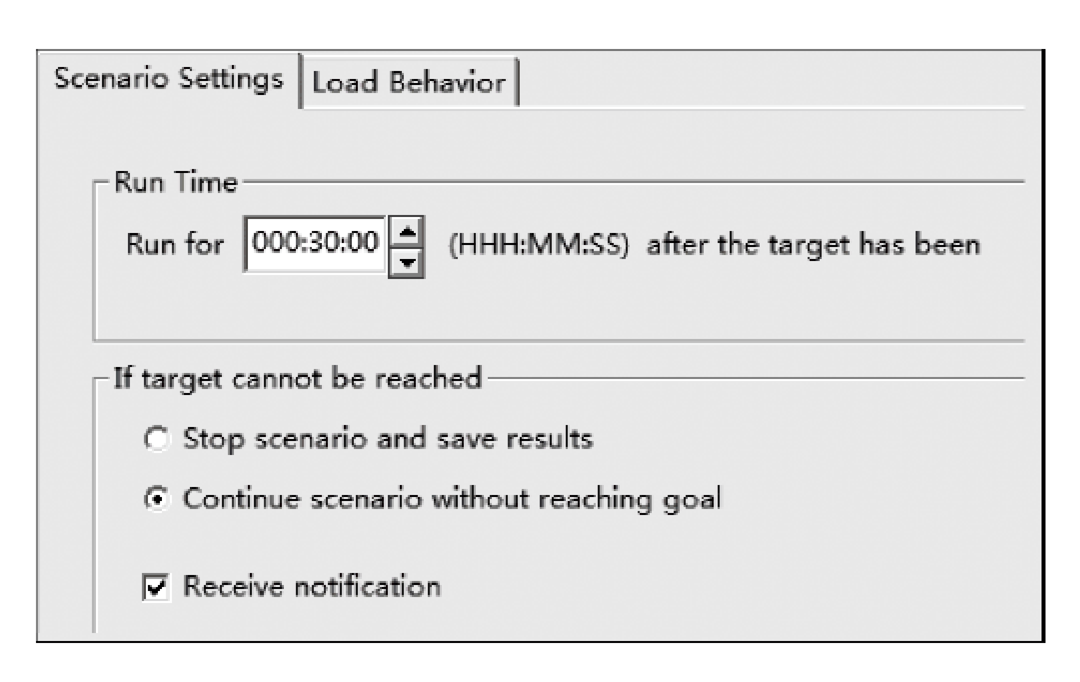


图 15-7　目标场景的场景设置

Run Time 是指当目标达到后，需要继续运行多长时间来测试系统的稳定性，默认情况为 30min。目标场景是定性型场景，目标达到并不代表系统就满足了用户需求，还需要进行一段时间的稳定性测试，确保该指标能够在一段时间内达到。

而如果目标无法达到，又该如何处理呢？

① Stop scenario and save results：如果无法达到目标，那么整个场景停止运行。

② Continue scenario without reaching goal：无法达到目标场景仍继续运行。

当勾选了 Receive notification 复选框时，一旦出现目标无法达到的情况，Controller 会弹出信息框，提示信息 The target you defined cannot be reached。

（2）Load Behavior（负载生成）提供了对目标场景负载生成方式 Ramp Up 的设置，如图 15-8 所示。

此处可使用自动管理，也可以手工设置一个需要达到的目标时间。

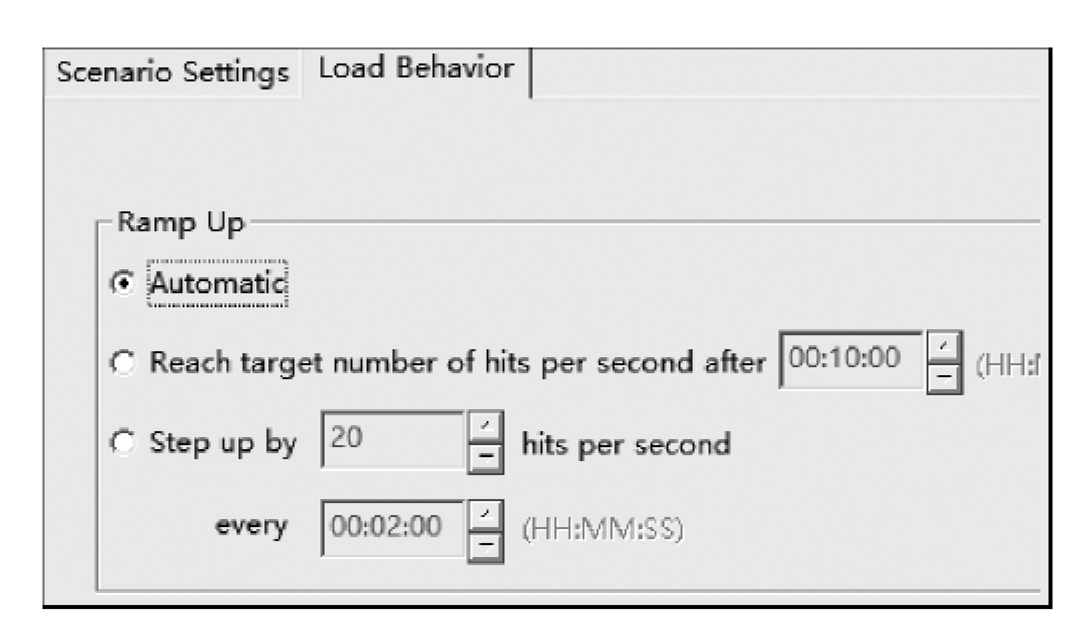


图 15-8　目标场景的负载规则设置

当设置完成后，就可以启动目标场景，Controller 会自动调整用户个数形成负载，确认在这种负载情况下，定义的目标都可以达到。

目标场景的目的就是通过设置目标来验证系统能否达到目标，在项目最后需要确认质量时可以使用目标场景来完成最终的测试报告，而当需要定位瓶颈的时候还是推荐使用手工场景。在最终的验收测试中经常需要多个功能同时满足某一性能目标。

2. 手工场景

手工场景(Manual Scenario)就是自行设置虚拟用户的变化，通过设计用户的添加和减少过程，来模拟真实的用户请求模型，完成负载的生成。

手工场景是定量型性能测试，通过掌握在负载的增加过程中系统各个组件的变化情况，从而定位性能瓶颈并了解系统的处理能力，一般在负载测试和压力测试中应用。

手工场景主要是在设计用户变化，通过手工场景可以帮助人们分析系统的性能瓶颈。

手工场景的核心就是设置用户负载方式，通过 Scenario Schedule 的 Schedule By 和 Run Mode 选项可以对虚拟用户的负载方式进行设置，如图 15-9 所示。

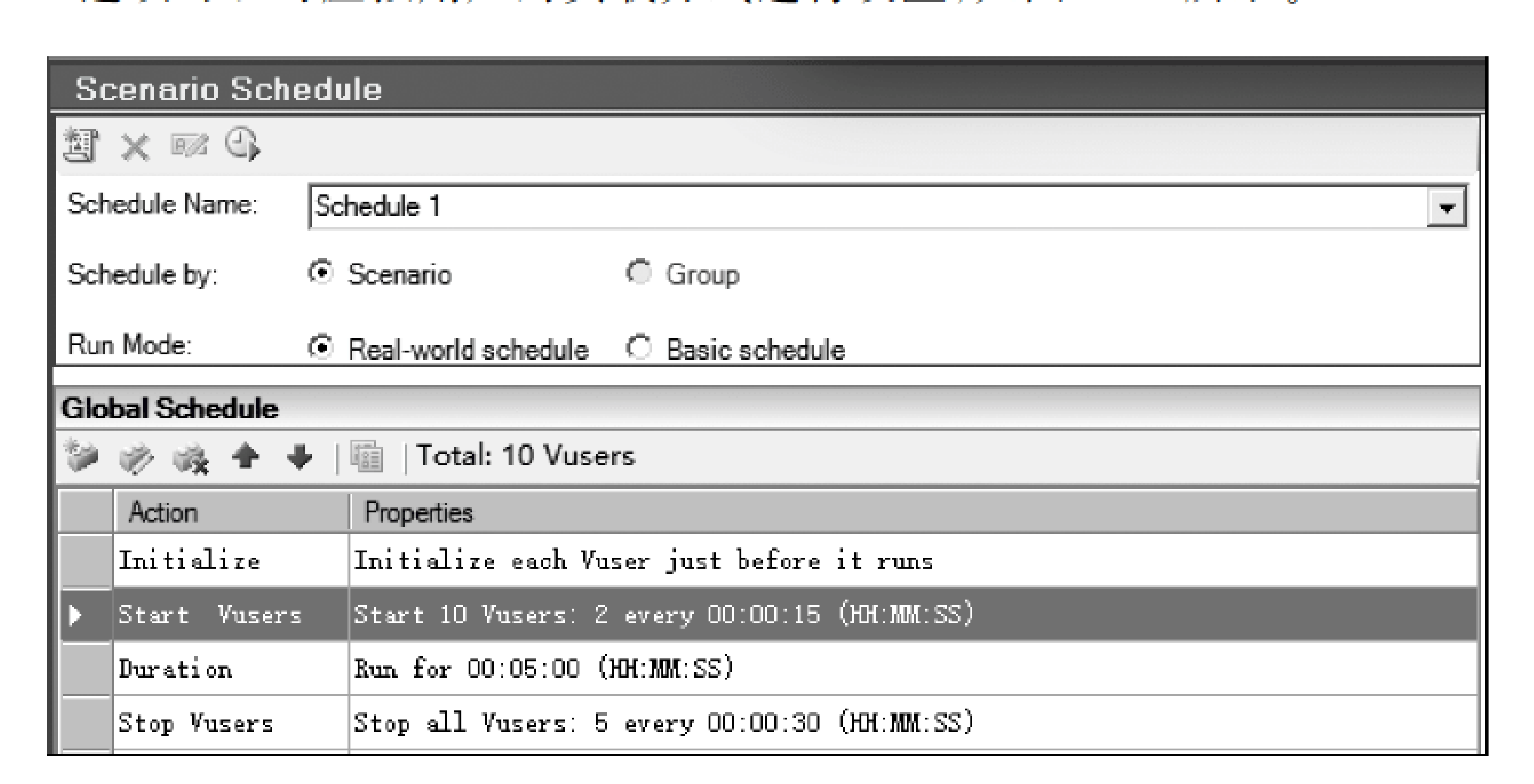


图 15-9　手工场景中的 Scenario Schedule

设置场景的最大在线用户为 10 人，每隔 15s 增加 2 个负载用户，1min 后达到最大在线用户数，持续 5min 后，用户每 30s 结束 5 个负载用户，整个场景共耗时 6min30s。

通过设置 Global Schedule 来设计用户的增加和减少过程，具体的用户负载情况会在右侧的 Interactive Schedule Graph 中显示出来，如图 15-10 所示。

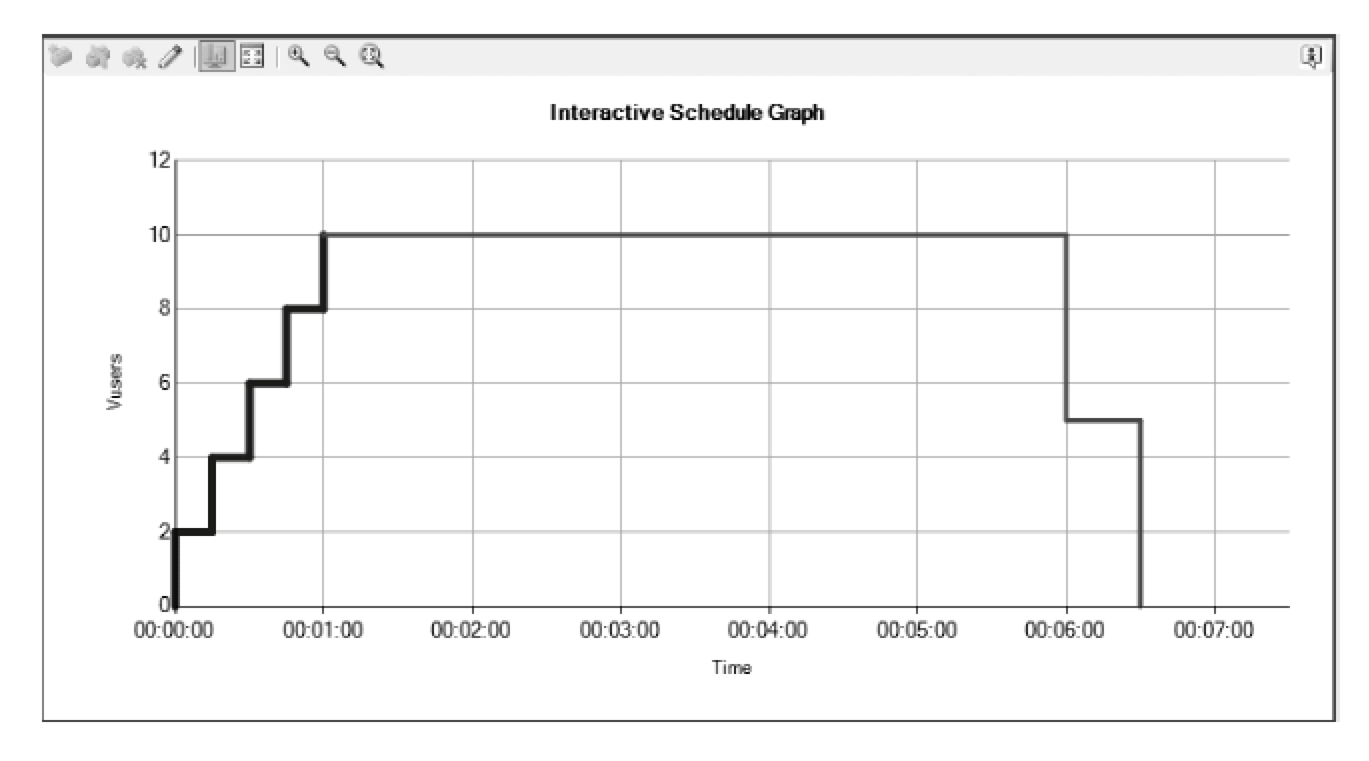


图 15-10　Interactive Schedule Graph 场景负载图

手工场景在 Schedule by 中分为 Scenario 模式和 Group 模式。

1）Scenario 模式

Scenario 模式是指所有脚本都使用相同的场景模型来运行，只需要分配每个脚本所使用的用户个数即可。

Scenario 模式下的 Run Mode 有两大类：Real-world schedule 和 Basic schedule。在 LoadRunner 9 以前的手工场景设计中，只有一种情况可以设定，就是一个用户脚本的运行场景分为用户上升、用户持续、用户下降三个过程，只要系统满足了峰值数据后，即可证明系统能够满足性能需求，但这并不是真实的负载情况。从 LoadRunner 9 开始提供了 Real-life schedule（在 LR11 中改名为 Real-world schedule），也就是说可以建立一个完全真实的场景，不用像以前那样只能模拟一座山峰，从而实现压力测试和真实情况下的负载测试。

① Real-world schedule（真实场景模式）。

和以前不同的是这里可以通过 Add Action 来添加多个用户变化的过程，包括多次负载增加 Start Vusers、持续时间 Duration 或递减 Stop Vusers，如图 15-11 所示。

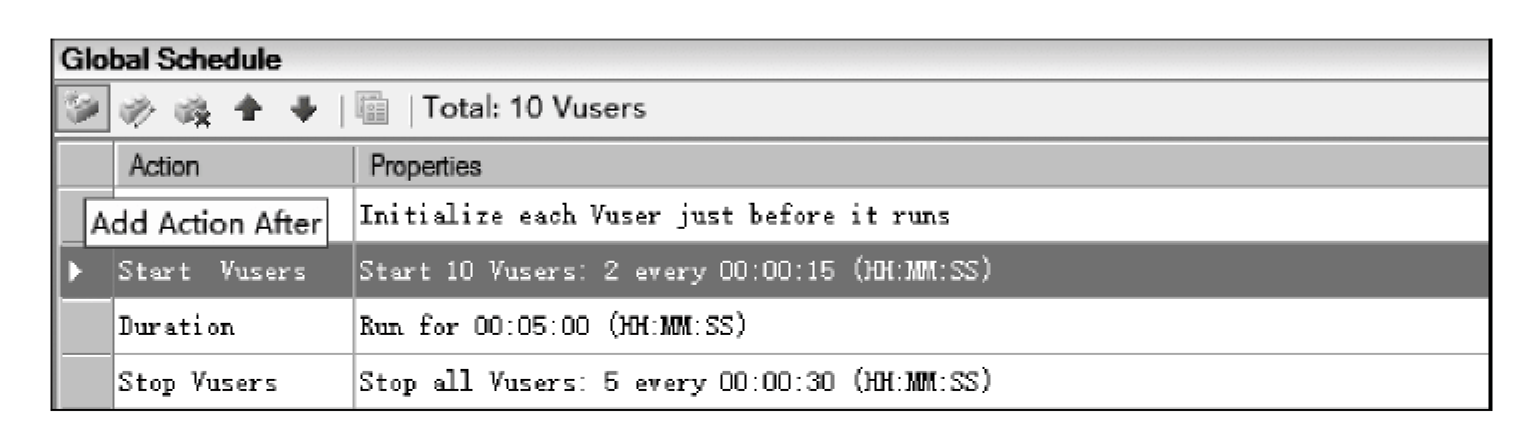


图 15-11　Real-world 下的 Global Schedule 场景添加负载策略

先来学习设置用户的初始化方式。双击 Initialize Action，弹出 Edit Action 窗口，如图 15-12 所示。

系统提供了三种初始化用户的方式，一般使用默认选项即在每个虚拟用户开始运行前进行初始化。

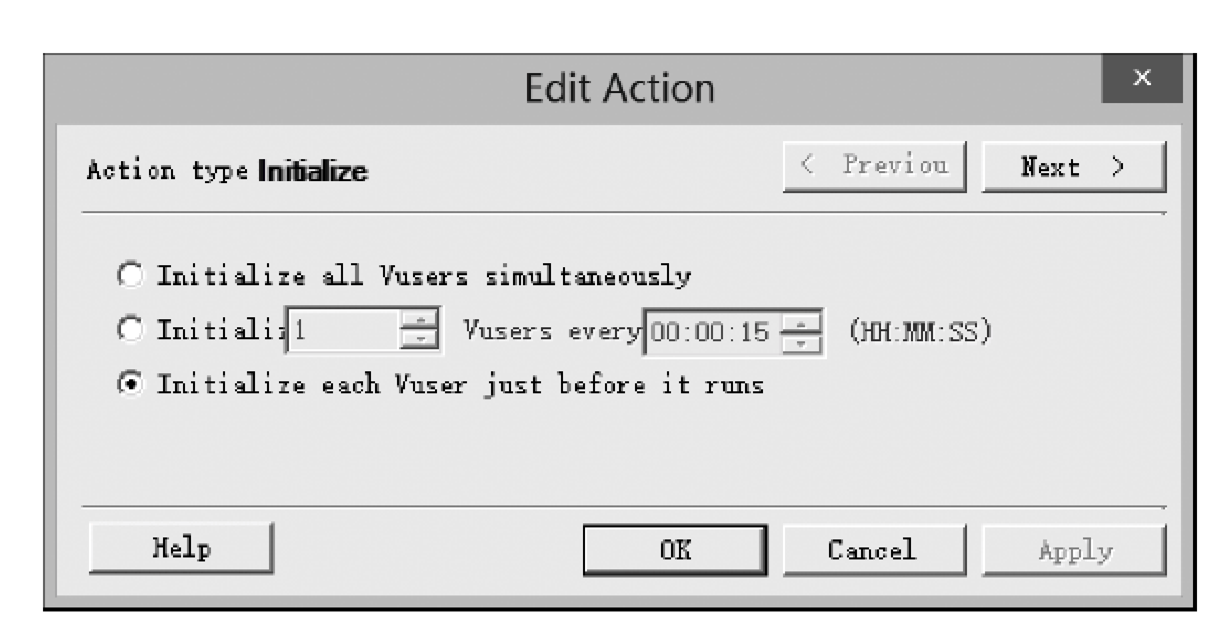


图 15-12 虚拟用户初始化策略

然后学习设置负载增加 Start Vusers。双击 Start Vusers 可以打开负载用户增加的策略设置对话框，如图 15-13 所示。

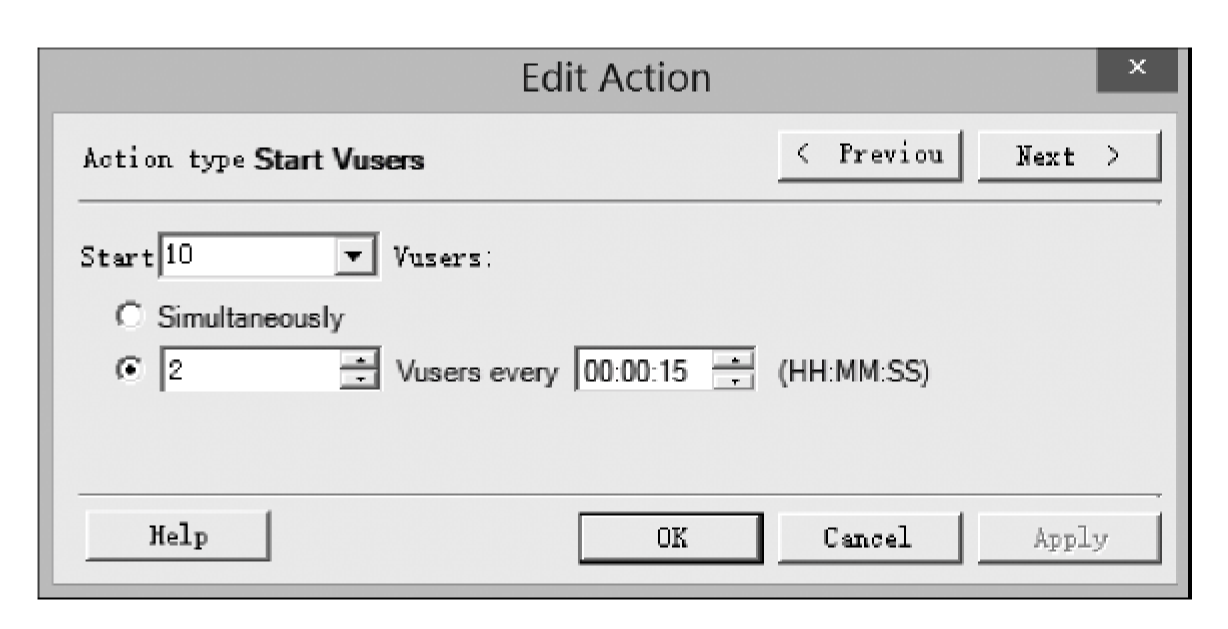


图 15-13 修改 Start Vusers 用户增长负载趋势

在这里可以设置产生负载的用户数，在默认情况下一般使用每隔一段时间增加一定的用户负载方式，但也可以设置为立即一次性加载用户。

建议设置为周期性负载增长模式，这样能够更加有效地获得系统在各个负载下的性能指标（避免一次负载太大，系统无法承受），除非需要做某些特殊情况的模拟。

接着设置负载持续时间 Duration。双击 Duration 打开设置窗口。在这里可以设置持续时间长度，通过一定时间的负载可以测试系统在该负载情况下的稳定性，也可以选择只执行一次脚本，如图 15-14 所示。

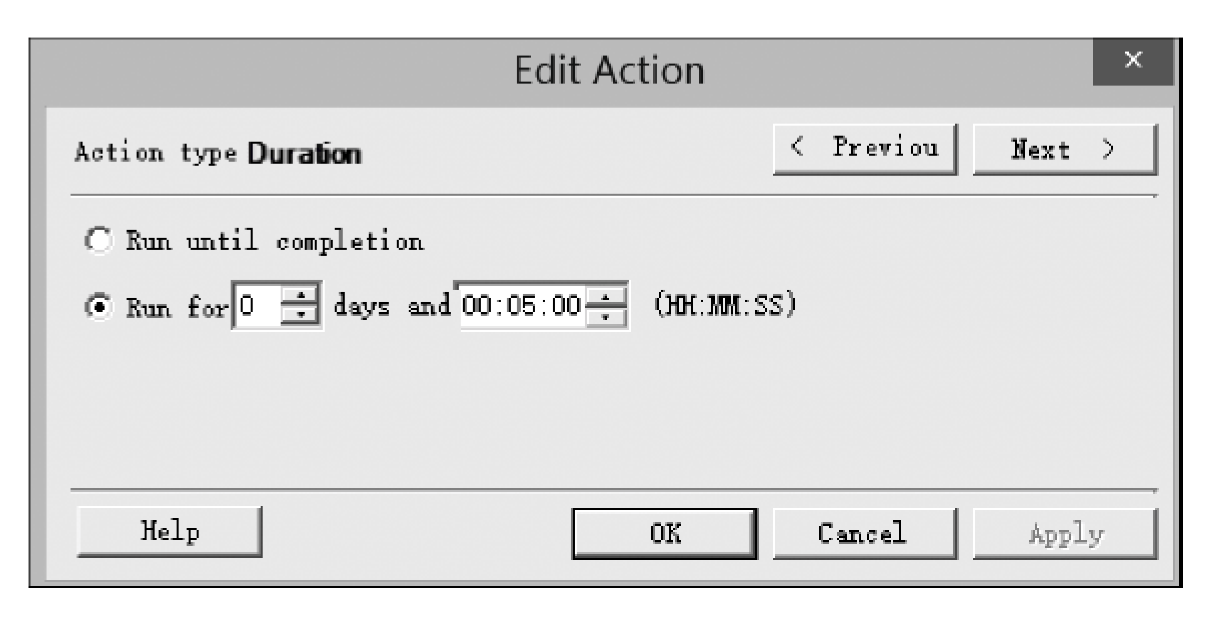


图 15-14 修改用户负载持续时间

最后设置负载释放的过程 Stop Vusers。双击 Stop Vusers 打开设置对话框，这里提

供了两种释放负载的策略，如图15-15所示。一般来说可以设置用户直接停止，也可以通过设置负载逐渐下降，分析系统回收资源的能力。

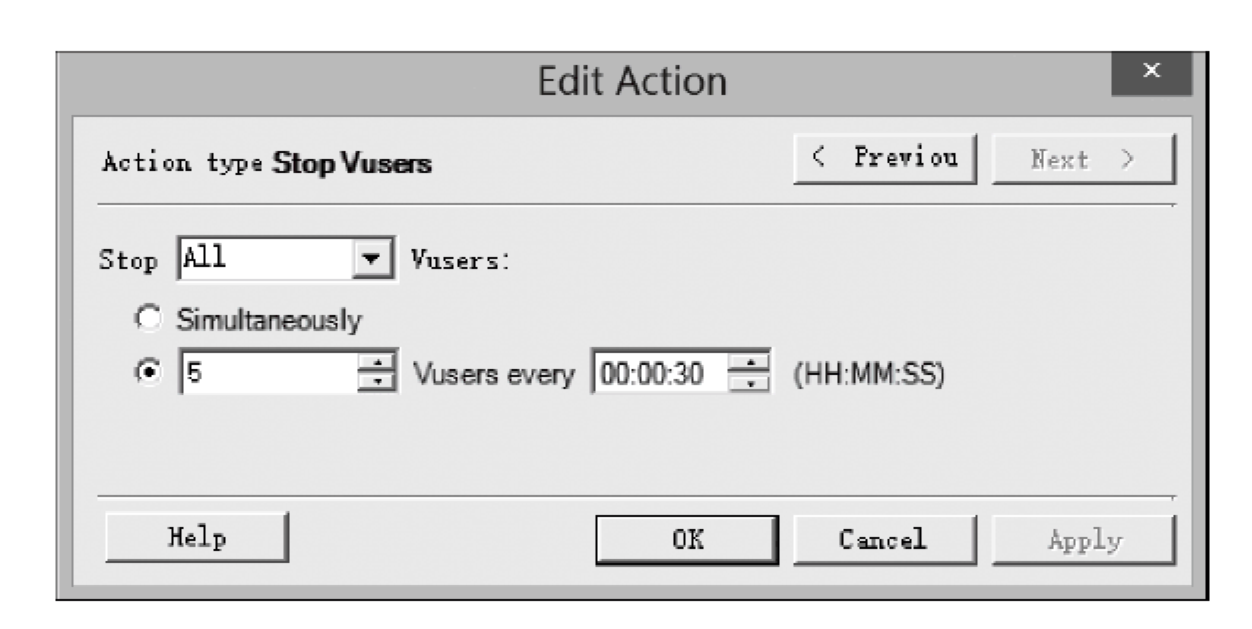


图15-15　修改Stop Vusers用户负载递减策略

通过反复添加Start Vusers/Duration/Stop Vusers可以生成一个波浪形的场景，正是因为这是一种完全自由的场景设计方式，所以才被称为Real-world，即完全真实地模拟用户负载的过程，通过这个过程的模拟克服了以前场景想要模拟负载反复起伏的困难。

Real-world schedule模式常常用在压力测试和稳定性测试中，了解系统在长时间波动负载下的资源管理能力，而Real-world的负载策略是根据性能需求模型来确定的。

② Basic Schedule(基础模式)。

这种模式就是老版本的场景设计模式，只能设置一次负载的上升持续和下降。常见的负载测试都是通过Basic方式实施的。

在Basic Schedule模式下，用户的Duration持续时间设置会多出Run indefinitely选项，是指脚本会永不停止地运行下去。

基础模式其实在很多时候已经够用了，通过它生成一个峰值负载，只要系统能够满足这个峰值即可。一般来说只要在峰值下满足性能需求，那么常规情况也能满足性能需求。

但是有时候会发现虽然峰值性能指标能满足，但系统还是会出问题。这是因为系统并不是长期都处在高负载状态下的，随着负载的变化，系统的资源在不断地申请释放。如果在这个过程中存在微量的资源回收失败，那么时间一长系统就会出问题。另一方面性能测试需要对用户行为进行模拟，如果场景只有经典模式那么如何模拟真实的用户负载波动呢？所以这个时候Real-world就有意义了，它可以设置一个与真实情况类似的场景来实现负载。

负载的真实性是受到脚本影响的，一般Real-world运行的脚本会更偏向于模拟用户操作流程，而Basic的脚本则更偏向于模拟一种操作。以上介绍了两种场景的运行模式，那么当多个脚本在场景运行时，如何配置它们之间的关系呢？

在Scenario模式下经常需要模拟多个脚本共同运行的情况，从而测试系统在多种业务下的处理能力。

在手工场景中，用户脚本都被称为Group，这是因为每一个用户组都代表一种脚本操作，通过组名来区别脚本之间的关系。

如何修改各个Group的Quantity用户数呢？首先可以在Start Vusers内修改开始的用户总数，然后需要将场景的用户修改为百分比模式，选择Scenario菜单下的Convert

Scenario to the Percentage Mode，将场景用户模式改为百分比模式，如图 15-17 所示。

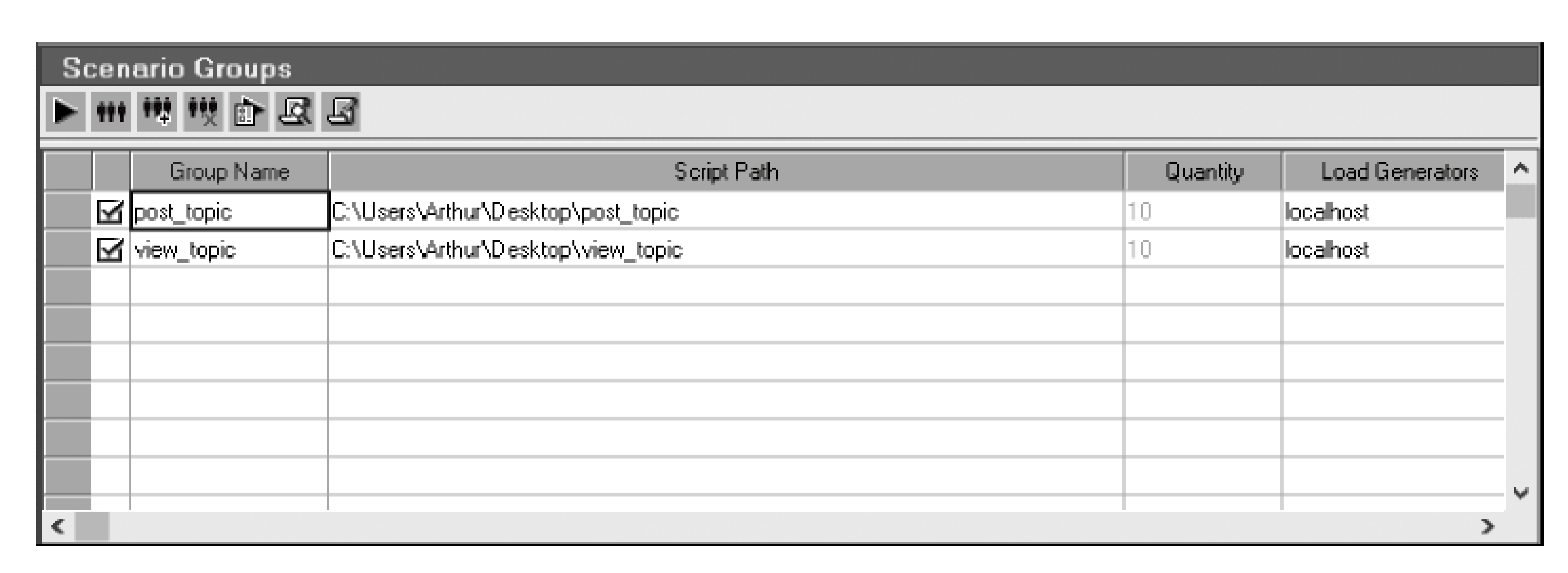


图 15-16 手工场景中添加多个 Group 脚本

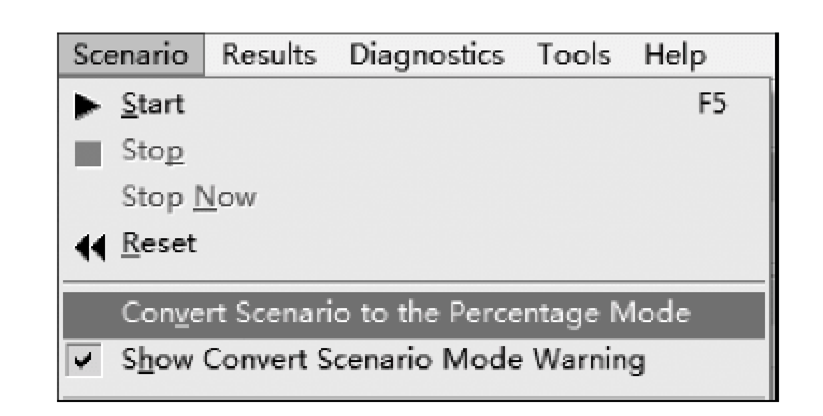


图 15-17 切换手工场景用户数为百分比模式

通过场景的运行图可以发现两个脚本是使用同样的负载方式进行的，只是根据用户的比例分配负载增加的趋势。

2）Group 模式

在 Group 模式下，除了可以独立设置脚本开始原则以外，还可以通过 Start Group 策略为脚本之间设置前后运行关系。

双击 Group Schedule 下的 Start Group Action，打开 Start Group 策略，设置该脚本在手工场景下的 Group 模式中如何开始运行，如图 15-18 所示。

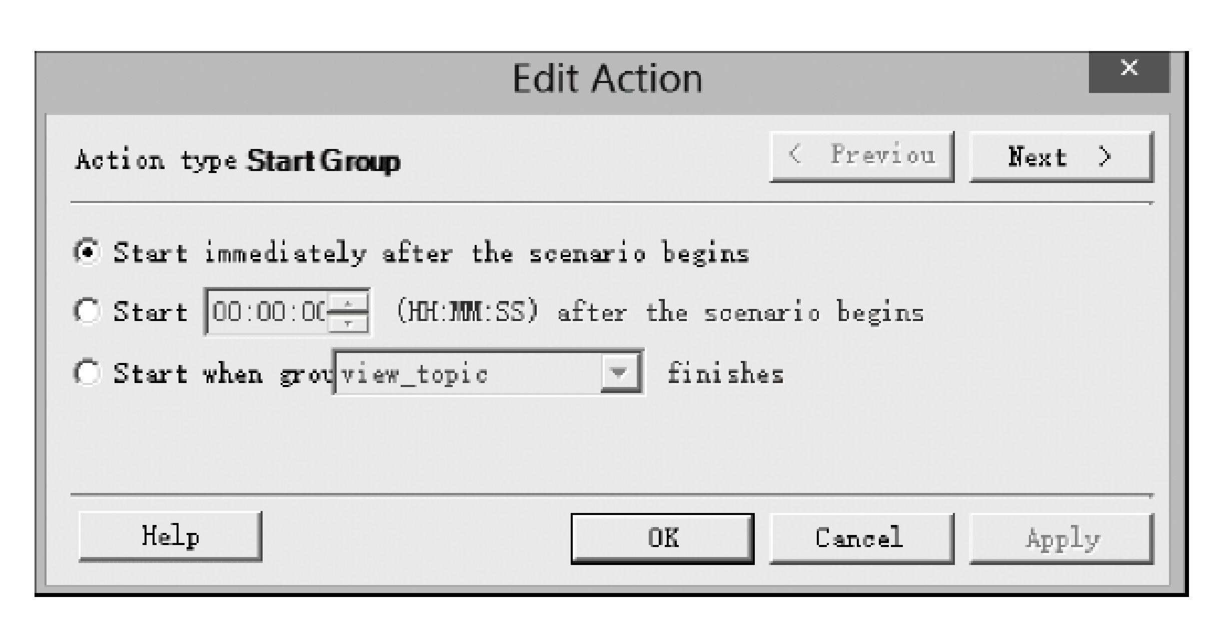


图 15-18 Group 模式下的 Start Group 策略

其中，Start immediately after the scenario begins 表示当场景一开始就立即运行；Start（HH：MM：SS）after the scenario begins 表示当场景运行多长时间后再运行；Start when group [] finishes 表示当某一个 group 结束后再运行。

脚本之间的场景设计使用不同的颜色区别，选中脚本可以修改该脚本的运行设置，如图 15-19 所示。

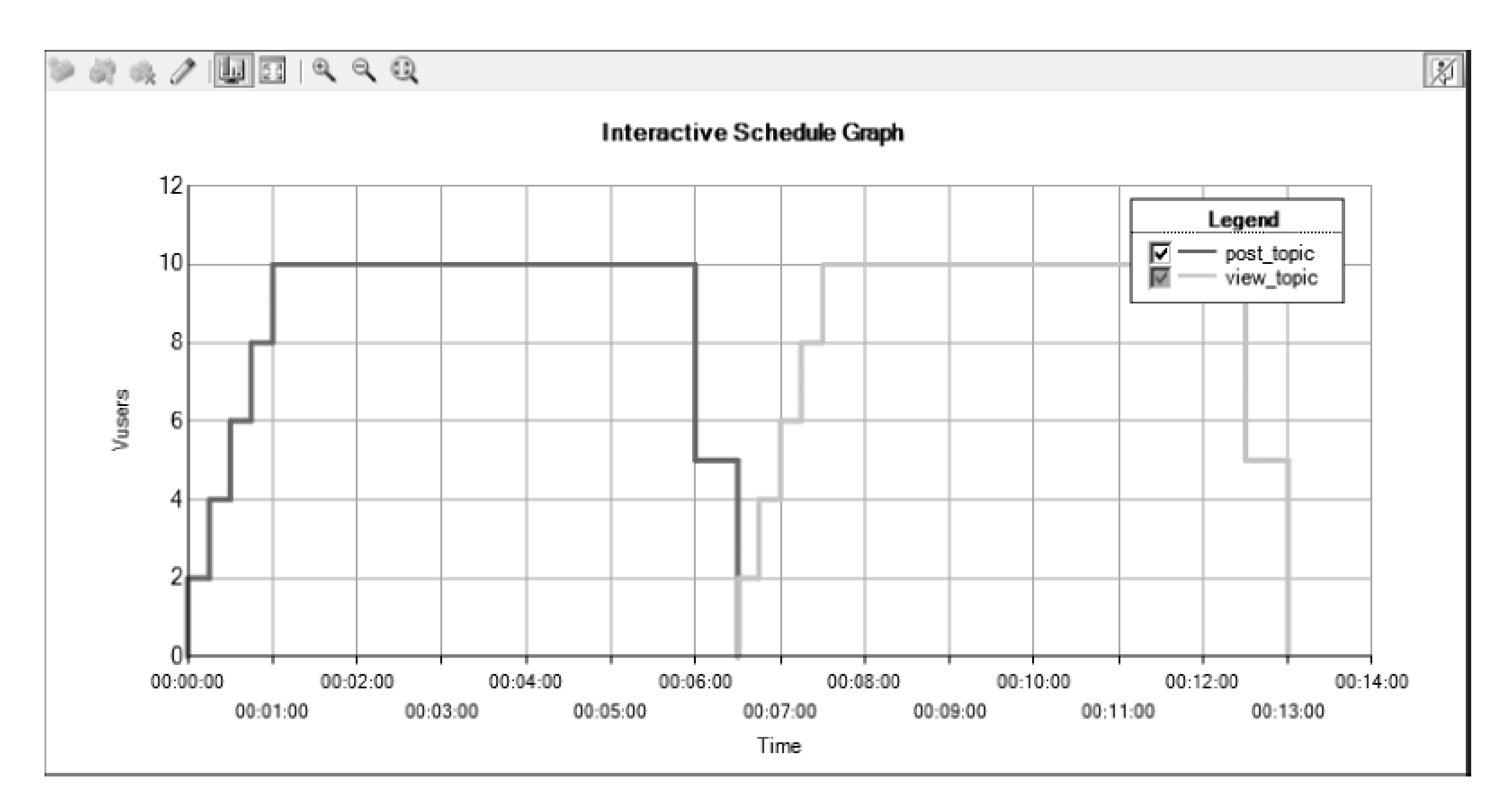


图 15-19 Group 模式下多脚本 Interactive Schedule Graph

这里设置的 view_topic 脚本是在 post_topic 脚本场景运行结束后再运行的。在某些负载策略中需要使用 Group 模式才能完成场景设计。

到这里介绍了目标场景和手工场景两种场景类型。场景提供脚本运行的方式，通过目标场景进行定性型负载，通过手工场景进行定量型负载。

设计场景在工具上并没有复杂的内容，关键在于性能需求和性能测试目标，设计的场景到底为了测试什么东西是在场景设计前需要好好考虑的。一般通过在场景中运行一种用户行为可以对某一个功能点进行性能测试和分析，如果需要对整个系统的运行情况进行性能测试和分析，就需要同时运行多个脚本。如果在场景中加载多个脚本，并分别设置其负载方式，就能完成真实情况下的负载模拟。

15.1.2 负载生成器管理

当对场景进行设计后，接着需要对负载生成器进行管理和设置。Load Generators 是运行脚本的负载引擎，在默认情况下使用本地的负载生成器来运行脚本，但是模拟用户行为也需要消耗一定的系统资源，所以在一台电脑上无法模拟大量的虚拟用户，这个时候可以通过调用多个 Load Generators 来完成大规模的性能负载。

Load Generators 的核心是 mmdrv. exe 进程，mmdrv. exe 负责运行脚本模拟用户行为，该程序支持进程或线程的方式，通过 Runtime Settings 即可进行设置。

打开 Scenario 菜单下的 Load Generators，如图 15-20 所示。

Load Generators 管理器列出了所管理的负载服务器列表，添加负载引擎只需要在这里单击 Add 按钮，然后在对话框中输入需要连接的负载引擎所在的电脑 IP 以及对应平台即可(确保对应平台上已经安装并启动了 Load Generator 服务)，如图 15-21 所示。

添加该引擎后，可以单击 Connect 按钮连接一下，如果出现 Ready 则说明正确连接，该负载生成服务器可以使用，否则就需要检查一下是什么原因导致的连接错误。

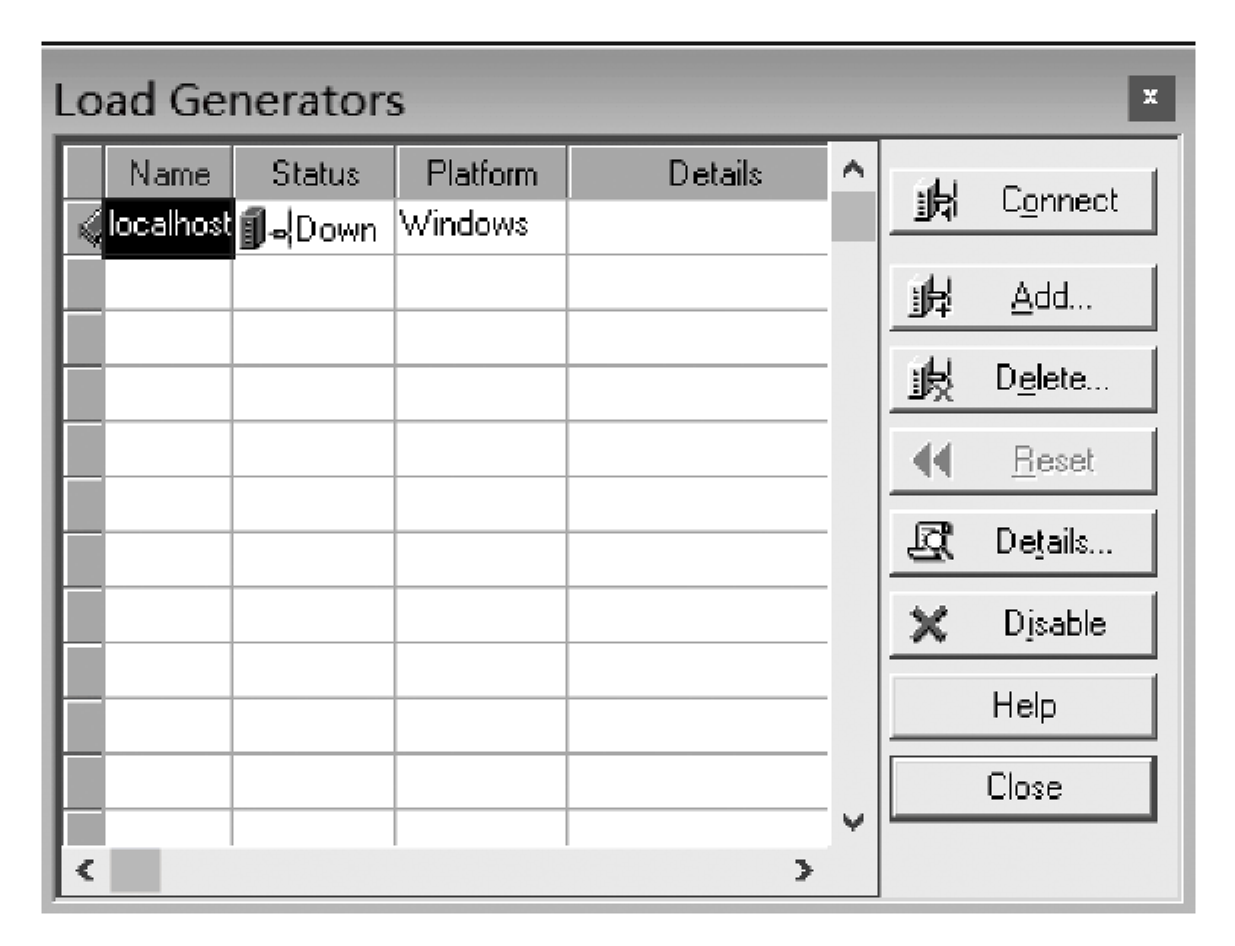


图 15-20 Load Generators 管理器

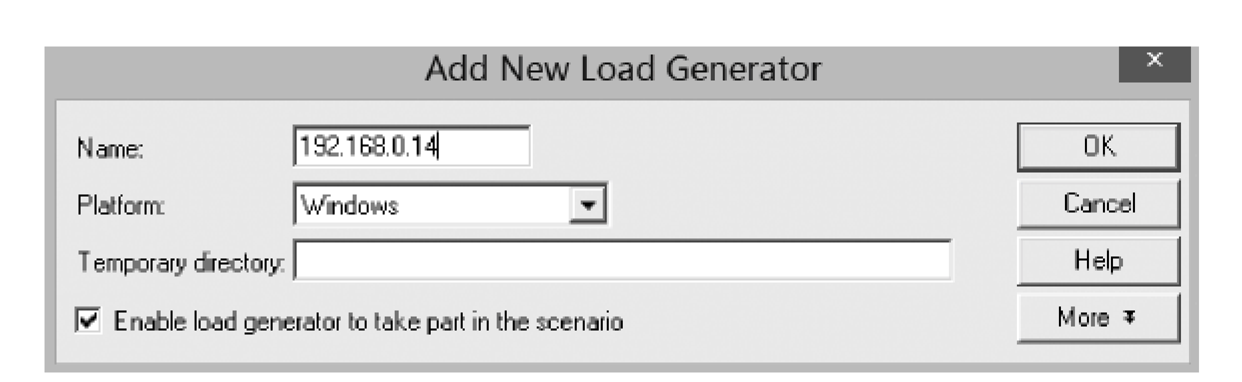


图 15-21 添加远程 Load Generators

当远程负载服务器被成功添加至负载服务管理器中后，就可以在 Scenario Group 中的 Group 脚本右侧选择使用哪一台负载服务器来运行对应的脚本了，如图 15-22 所示。

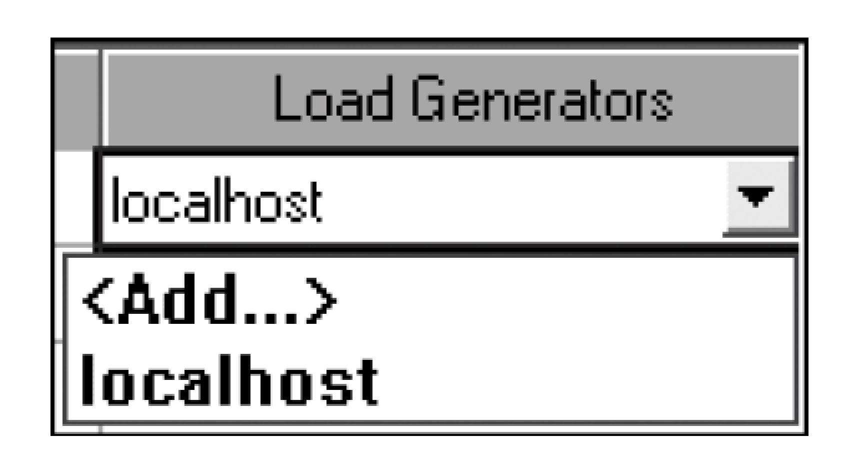


图 15-22 设置脚本运行所在的负载生成器

通过设置多个 Load Generator 可以有效地增加负载量，解决单台电脑无法模拟大量负载的问题。当场景开始运行时，Controller 会先将脚本传输到各个负载生成器上，等到运行结束后，各个负载生成器的日志会被 Controller 回收。在大多数情况下，使用进程方式时一个 Vuser 会占用接近 3MB 的内存，而使用线程方式时一个 Vuser 大概只占用 200KB 的内存。为了保证负载生成的有效性，请在真正实施性能测试前先测试一下负载器是否存在硬件瓶颈(生成负载时的 CPU、内存、带宽占用情况)，确保负载生成时负载器自身不会成为瓶颈，其 CPU 和内存的使用率最好不超过 80%。

15.2 场景执行

15.2.1 场景运行的准备工作

1. 用户管理

在 Scenario Groups 工具条中，除了运行脚本按钮、查看脚本和 Run-Time Setting 功能外，Virtual Users 管理器是经常使用的一个功能，该功能提供了一个对负载用户进行快捷有效监控的操作平台，如图 15-23 所示。

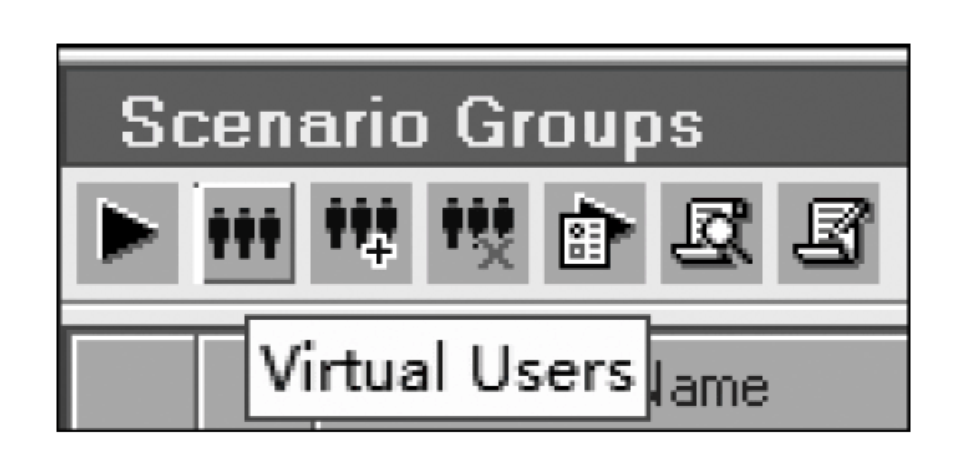


图 15-23 Scenario Groups 工具条

单击 Virtual Users 按钮，弹出虚拟用户管理器，如图 15-24 所示。

Vusers(20)

post_topic　All Vusers

ID	Status	Script	Load Generator	Elapsed Time
1	Down	post_topic	localhost	
2	Down	post_topic	localhost	
3	Down	post_topic	localhost	
4	Down	post_topic	localhost	
5	Down	post_topic	localhost	
6	Down	post_topic	localhost	
7	Down	post_topic	localhost	
8	Down	post_topic	localhost	
9	Down	post_topic	localhost	
10	Down	post_topic	localhost	

Run　Gradual Stop　Stop　Reset　Details...　Add Vuser(s)...　Help　Close

图 15-24 虚拟用户管理器

在场景运行前可以设置待运行虚拟用户的状态，如手动启动用户执行，也可为场景添加或停止用户。

当场景运行时，可以通过该功能对某个正在运行的用户进行监控。例如：选择某个正在运行的用户，右键菜单打开 Show User 功能，可以获得该用户的运行状态，也可以选择 Show Vusers Log 功能打开该用户运行的日志，还可以通过过滤规则来查找不同状态的虚拟用户，也可以通过函数动态为虚拟用户管理器中的 Status 添加信息。

2. 运行设置

在场景运行前还需要对脚本的运行策略进行设置，确保整个场景中所有用户的运行方式正确。注意在 Controller 中 Run-Time Setting 独立存放在场景.lrs 文件中，并不会影响该脚本在 VuGen 中运行的设置。

在场景运行前应该对以下选项进行检查设置，以确保脚本正确执行。

① Think Time。

在 VuGen 中 Think Time 默认为忽略，但是在场景中，该选项会自动按照脚本录制的 lr_think_time 函数进行运行。Think Time 可以模拟真实用户的操作等待，如果该时间设置得太短，那么得出的性能数据就会比较悲观(模拟的用户操作比真实用户快，服务器的负载压力会比正常情况大，从而结果较差)，反之结果会过于乐观，所以这个时间不能随意设置。由于录制脚本的时候对业务比较熟悉，所以会导致 Think Time 较小，这里可以尝试取一个熟练用户的操作速度和一个新用户的操作速度的平均值来设置合理的 Think Time 值。

② 场景中 mmdrv.exe 负载的生成方式。

Load Generators 会调用 mmdrv.exe 来生成负载，而负载的生成分为进程方式和线程方式。

使用进程模式模拟负载的资源开销会相对较大，每个虚拟用户会使用一个单独的 mmdrv.exe 来完成负载的实现，这样做用户之间会相互独立，不会相互影响。

而如果使用线程方式，那么所有用户都是在一个 mmdrv.exe 上模拟的，用户行为使用线程方式，模拟消耗的资源较小。

一般来说使用线程可以在固定的硬件平台上产生更多的负载模拟，但使用线程也会存在不稳定的情况，导致用户脚本执行错误。

③ 系统日志设置。

在场景中系统日志会从 Always send messages 变为 Send messages only when an error occurs，不出现错误就不记录日志，这样可以减少负载时记录日志的资源开销，从而提高模拟效率。当需要进行错误跟踪时，再将其打开。

④ 自动化事务。

在脚本中都会对关键的操作添加事务从而获得响应时间，在 LR11 中默认没有打开自动化事务，所以对比以前版本如果没有设置手工事务默认在场景中将无法看到事务时间。如果在这里需要自动补充事务时间，可以在场景运行时设置该选项。

⑤ 带宽模拟。

带宽会直接影响到事务的响应时间，而真实环境下每个用户的带宽也是有限的，这里需要为用户设置一个合理的带宽来得到真实用户访问的响应时间。

通常情况下一个客户端在访问一个 Web 网站时的平均连接速度是 30～50kB/s，这里可以选择 512kb/s(DSL)为场景中的每个用户分配 512kb/s 的带宽。为了避免出现由于模拟用户过多，导致 Load Generator 上出现带宽瓶颈的情况，需要在设置前进行计算。如果设置每个用户使用 512kb/s 的带宽，那么在 100Mb/s 的总带宽下，最多模拟 200 个

用户。

⑥ 集合点策略。

集合点是业务操作上的一个操作聚集点。如果脚本中含有集合点,则需要根据需求对集合点的策略进行设置。当场景需要多脚本并发负载时,只需要设置同名集合点即可实现。

15.2.2 有效的场景运行技术要点

场景运行时,负载生成器在运行过程中:保存负载生成器各自的运行过程中的结果;在校本执行结束以后将相关的执行结果数据传送回控制台以便分析。控制台在运行过程中:保存事务以及系统监控的各种数据;通过集合点方法去控制虚拟用户的同步(可选);收集各个虚拟用户的错误与确认信息。

在执行场景的过程中,团队合作分工如下:

(1) 测试工程师通过 LR 控制台监控事物处理性能和服务器的情况;

(2) 网络工管理员在负载情况下监控网络性能;

(3) 应用管理员或者数据库管理员在负载情况下监控远程系统性能。

在有效地运行场景时,执行过程如下:

(1) 调试运行(Debug Run)。

目标:检验参数化的数据在并发情况下使用是否正确;

运行设置:选择扩展日志,选择接收服务器返回信息,忽略思考时间;

期待结果:运行过程中没有错误,这表明已经准备好继续下面的性能测试条件。

(2) 隔离大事务(Isolate Top Time Transactions)。

目标:挖掘在高负载下任何潜在的性能瓶颈;

运行设置:打开标准日志,忽略思考时间;

期待结果:挖掘性能最差的事务,为完全负载做最后各个方面的性能调优。

(3) 完全负载运行测试。

目标:模拟与现实相符的场景校验系统是否满足预期目标;

运行设置:关闭日志,打开思考时间;

期待结果:根据测试结果校验是否满足测试目标。

(4) 预测。

目标:在不添加资源的前提下校验系统未来可能出现的瓶颈;

运行设置:关闭日志,打开思考时间;

期待结果:负载测试目标应该被超过,响应时间达到 2 倍完全负载情况下的响应时间。

15.3 性能监控

15.3.1 性能参数监控方法

在场景运行前还需要对系统进行监控，记录负载生成的规则、负载数据和系统在负载下的资源使用情况。除了对负载的生成状态进行监控外，被负载的系统资源也需要进行监控。从某些角度来说性能测试的核心可能是监控而不是负载和调优。

LR 提供了各种性能监控指标来挖掘性能瓶颈，主要有以下几种监视器用于监控。

(1) "运行时"监视器：显示参与运行场景的 Vuser 数，Vuser 状态以及 Vuser 生成的错误数和类型。

(2) "事务"监视器：显示场景运行时各事务速率和响应时间。

(3) "Web 资源"监视器：监视场景运行期间 Web 服务器上的信息，主要包括 Web 连接数、吞吐量、HTTP 响应数、服务器重试次数和下载到服务器的页面数等信息。

(4) "系统资源"监视器：主要是监视场景运行期间 Windows、UNIX、Tuxedo、SNMP、AntaraFlameThrower 和 SiteScope 资源的使用情况。

(5) "网络延迟"监视器：显示关于系统网络延迟的信息。

(6) "防火墙"监视器：用来量度场景执行期间防火墙服务器信息统计的情况。

(7) "Web 服务器资源"监视器：用来量度场景运行期间 Apache、Microsoft IIS、iPlanet(SNMP)和 iPlanet/Netscape Web 服务器的统计信息。

(8) "Web 应用程序服务器资源"监视器：量度场景运行期间应用程序服务器 Ariba、ATGDynamo、BroadVision、ColdFusion、Fujitsu INTERSTAGE、iPlanet (NAS)、Microsoft ASP、Oracle9iAS HTTP、SliverStream、WebLogic(SNMP)、WebLogic(JMX) 和 WebSphere 统计信息的情况。

(9) "数据库服务器资源"监视器：用于量度场景运行期间数据库 DB2、Oracle、SQL 服务器和 Sybase 统计信息的情况。

(10) "流媒体"监视器：用来量度场景运行期间 RealPlayer 和 Media Player 客户端以及 Windows Media 服务器和 RealPlayer 音频/视频服务器的统计信息。

(11) "ERP/CRM 服务器资源"监视器：用来量度场景运行期间 SAP R/3 系统、SAP Portal、Siebel Server Manager、Siebel Web 服务器和 PeopleSoft(Tuxedo)服务器的统计信息。

(12) "Java 性能"监视器：用于量度 J2EE 对象级 J2EE 和 EJB 服务器对象的统计信息。

(13) "应用程序组件"监视器：用来量度场景运行期间 Microsoft COM+ 和 Microsoft.NET CLR 服务器的统计信息。

(14) "应用程序部署解决方案"监视器：用来量度场景运行期间 Citrix MetaFrame XP 和 1.8 服务器的统计信息。

(15) "中间件性能"监视器：用来量度场景运行期间 Tuxedo 和 IBM WebSphere MQ

服务器的统计信息。

(16)"基础结构资源"监视器：用来量度场景运行期间网络客户端数据点的统计信息。

15.3.2 根据测试目标添加性能监控参数

1. 设置监控属性

选择 Tools→Options 打开属性对话框，监控 Monitor 属性页如图 15-25 所示。

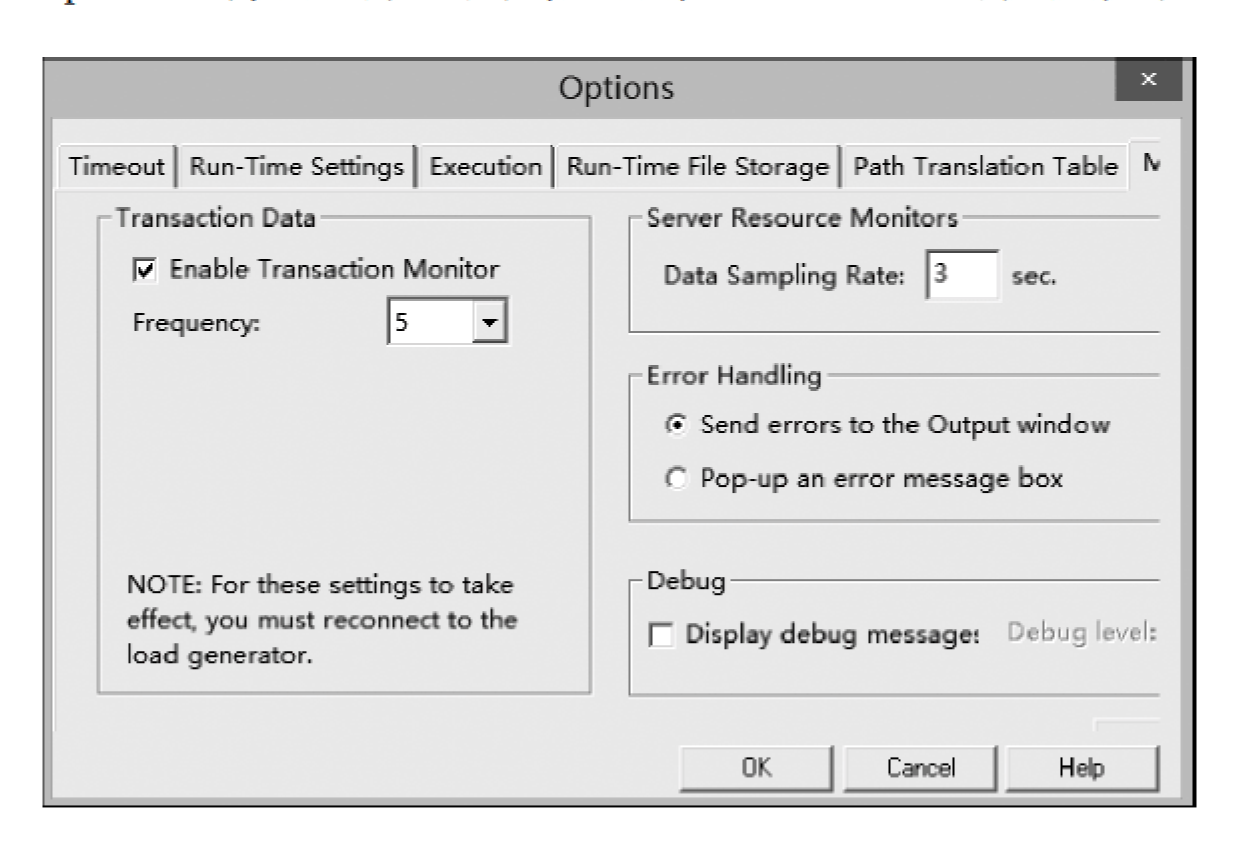


图 15-25 设置监控属性

选中允许事务监控复选框，并设置监视器向 Controller 发送更新的频率；同时选择错误处理选项，推荐为 Send errors to the Output window。

2. 配置监视器

启动 Monitor Configuration，进入监视器配置对话框，如图 15-26 所示。

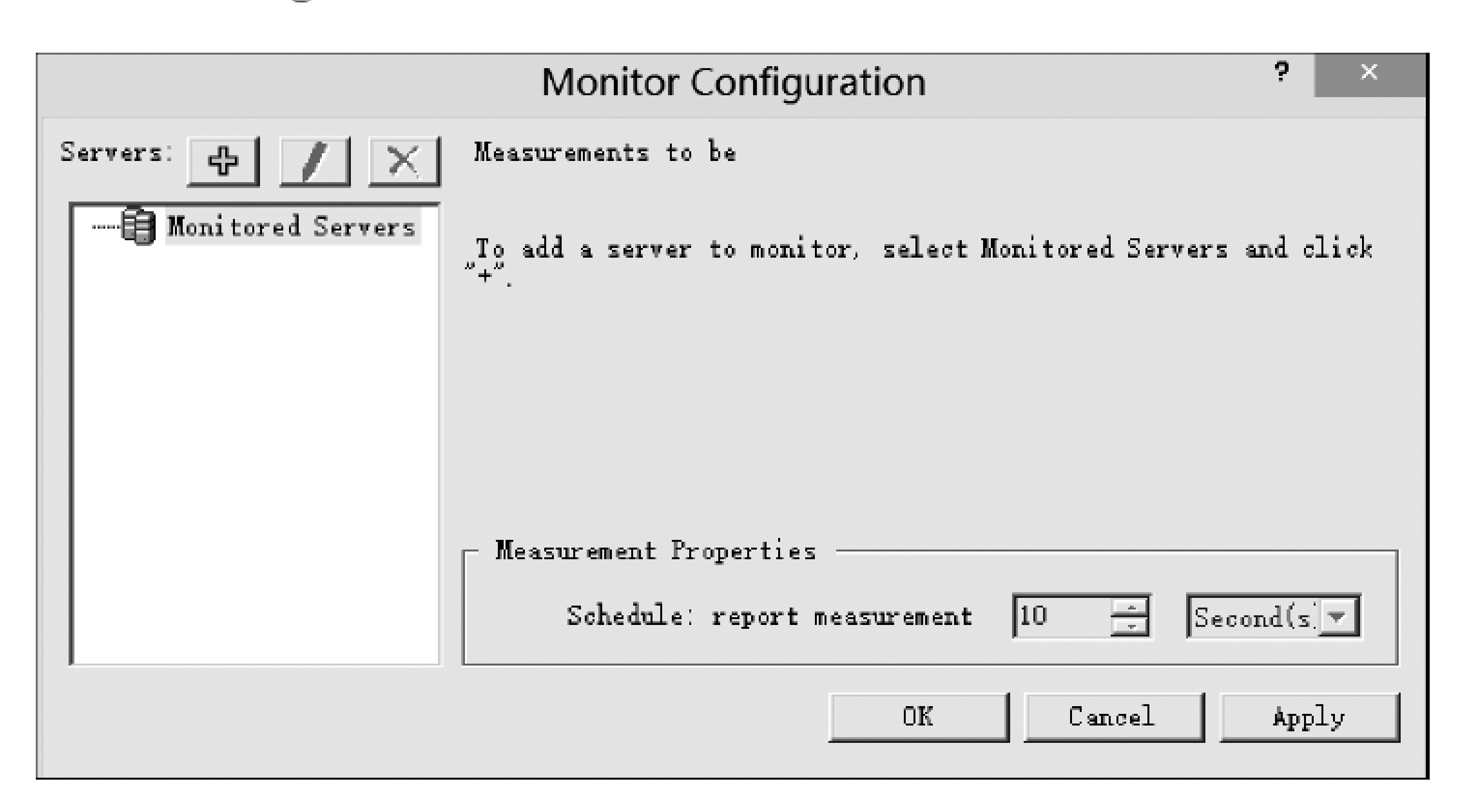


图 15-26 监视器配置对话框

单击左上角的+号按钮添加监视器，如图 15-27 所示。

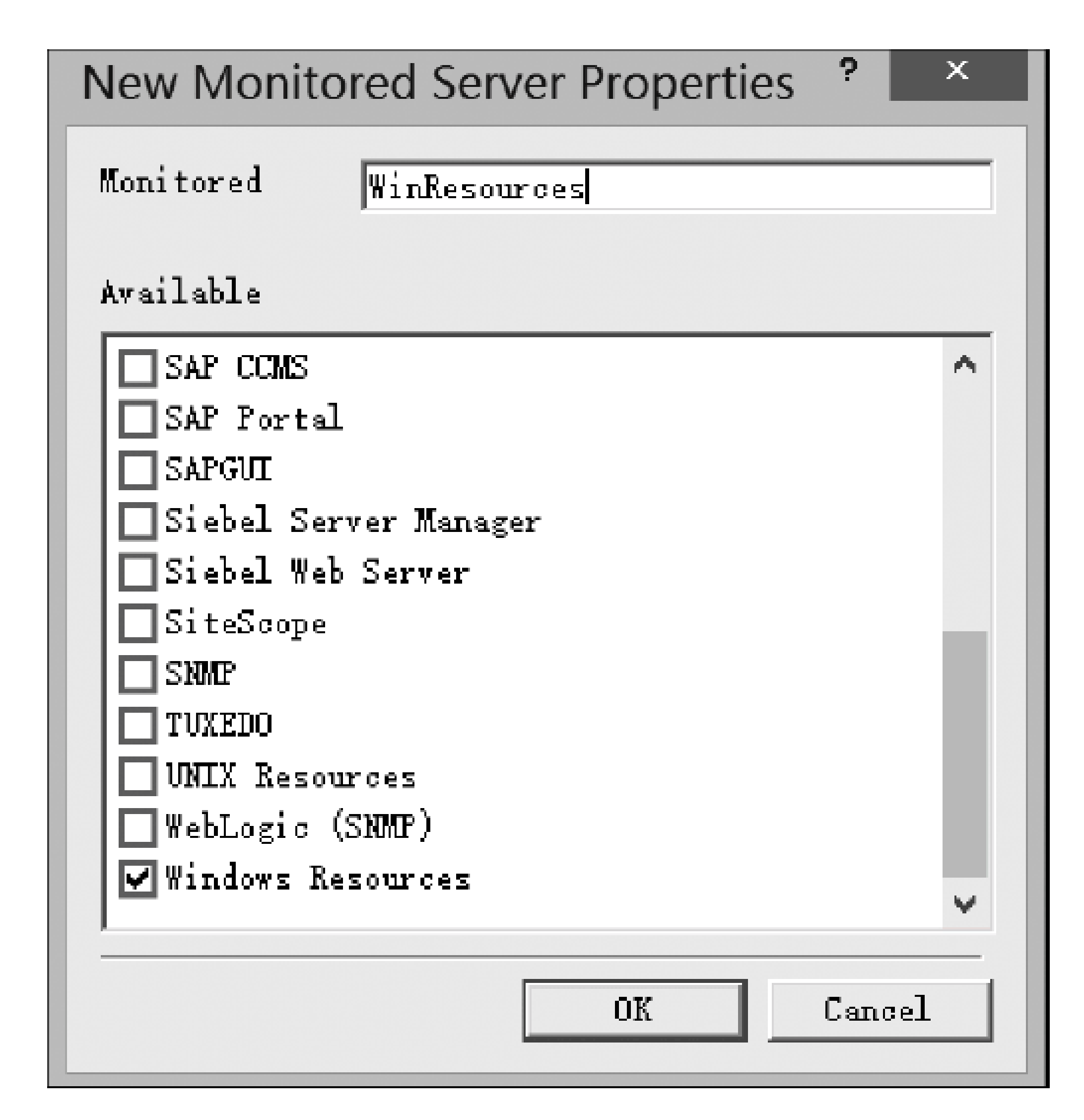


图 15-27 新增监视器

选择需要的服务器并设置其名称完成配置。

15.4 本章小结

场景就好比一个舞台,性能测试工程师就好比导演,一个个虚拟用户脚本就好比演员,通过情节设定(需求),将一个个演员放在这个舞台上进行演出(负载),而场下的观众(监视器)会对每个演员的表演进行评分(监视结果),最终确定最佳演员和最差演员(瓶颈)。场景运行是为了针对某一个功能或者在某种情况下对负载进行模拟,从而了解在这种负载下的系统情况。做一个演员容易,做一个导演就没那么简单了,性能测试工程师首先要有能力通过需求分析来实现场景的设计工作,其次需要对被负载系统的各个环节都有一定的了解和评论能力,甚至需要一个专家团队来协助分析,最终生成测试结果。

第 16 章　测试分析技术

在测试场景执行完成后，很多测试工程师认为最困难的阶段到来了——性能测试结果分析。因此，本章似乎很自然地就成为最重要的一章，但作者却认为性能测试分析并不是最难的工作。所谓“万丈高楼平地起”，也就说明性能分析的准确性同样取决于此前所做的设计与实施等“地基”是否可靠。因此可以说，性能测试分析仅仅是百米赛跑中最后的 20m 而已。当然，这并不是说性能测试分析不重要，因为“最后冲刺的 20m 没有跑好”，前面的工作做得再好也是徒劳。由此不难理解，性能测试分析工作开展的根基就是前面测试场景的执行结果。要想保证性能测试分析的结论是正确的，那么测试结果数据首先就应该是正确的，而这也意味着测试场景以及测试执行过程都应该是正确的。

实际上，性能测试从始至终都应该是相当严谨的一项工程，各个阶段的工作环环相扣，因此，性能测试工程师应该认真对待每一个阶段的工作。如果一味地追求找出系统瓶颈，无疑是舍本逐末的做法。

本章的主要内容如下：

（1）如何分析性能测试结果；

（2）如何从分析图中发现问题；

（3）分析图的处理方法；

（4）Analysis 分析报告。

16.1　分析性能测试结果

1. 判断测试结果是否有效

在 Controller 执行的测试场景结束后，首先要做的是判断采集到的结果数据是否真实有效。多数性能测试场景都需要迭代地进行测试，因此很多测试结果本身就不能反映真正的问题，而深入分析这样的结果纯属浪费时间。在本书中，主要探讨如何针对有效的测试结果数据进行分析。

判断测试结果是否有效，通常按下面的步骤进行。

第一步：在整个测试场景的执行过程中，测试环境是否正常。如果在测试过程中出现过异常，那么得出的结果往往不准确，无须进行分析。

例如，在测试执行过程中，测试机的 CPU 利用率经常达到 100%、测试环境的网络不稳定、一些系统参数配置不正确等，这样得出的测试结果没有必要进行分析，应该重新设置测试场景或调整测试环境，再次执行测试。

第二步：测试场景的设置是否正确、合理。测试场景的设置是否正确对测试结果有很大的影响。因此，当测试出现异常时，要对场景设置进行分析。

一些新手在使用 Controller 执行测试时，可能会同时在一台 PC 上加载全部虚拟用户——例如同时加载 1000 个虚拟用户，如果客户端来不及处理，就会有很多虚拟用户因不能初始化而失败。失败的根本原因不是被测试的应用服务器不能处理，而是压力根本没有传输过去。正确的做法是增加更多的 Generator 或逐步加压，使测试场景运行起来。

第三步：测试结果是否直接暴露出系统的一些问题。对测试场景的整个执行过程，没有必要对压力下系统运行正常的结果进行分析，因为这样的结果不能反映出系统的性能问题，应该进一步调整场景（如增大压力）进行测试。在测试过程中使系统表现不正常的测试场景生成的结果则要进行深入分析。实际上，分析能够反映性能问题的测试结果才是性能分析阶段的主要工作。

测试结果直接暴露系统存在性能问题的情形很多，例如在测试过程中一些用户事务的响应时间过长、系统支持的最大并发用户数过低、系统的应用服务器 CPU 利用率过高或内存不足等。对这类测试结果，性能测试人员需要借助 Analysis 对其进行深入分析，以发现一些潜在的性能问题。

2. 性能分析的基本原则

确定测试结果有效之后，接下来就要开始对测试数据进行深入的挖掘了。面对测试工具产生的纷繁复杂的原始测试数据，如何来进行分析呢？一个普遍遵循的原则是“由外而内，由表及里，层层深入”，如图 16-1 所示。

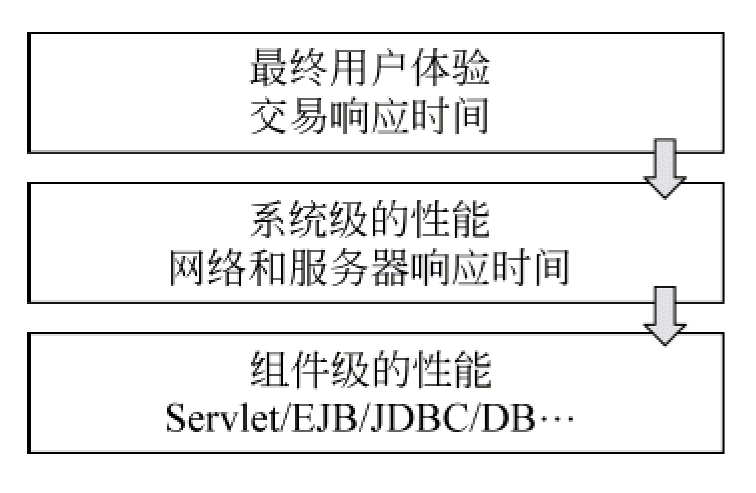


图 16-1　性能分析原则

对于一个应用系统，性能开始出现下降最直接的表象就是系统的响应时间变长。于是，系统响应时间成为分析性能的起点。性能分析的原则如图 16-1 所示，首先应该从原始测试数据中查看系统响应时间，判断它是否满足用户性能的期望。如果不能满足，则说明系统的性能出现了问题。发现系统存在问题后，就要判断系统在哪个环节出现了瓶颈。

现在的 IT 系统架构极其复杂，任何一个环节出现瓶颈，都会导致系统出现性能问题。要准确地判断瓶颈在什么地方，的确是一个棘手的问题。不过，任何复杂的系统都分为网络和服务器两部分。因此要考察的第二个问题就是：系统的瓶颈是出现在网络环节，还是服务器环节？

如图 16-2 所示，用户从客户端发起的请求数据包经过网络，传递到应用服务器，最后到达数据库服务器，服务器处理完毕后按原路返回客户端。在这个处理过程中，可以把整个时间分为两段：一段是 T_n，即网络的响应时间；另一段是 T_s，即服务器的响应时间，包括应用服务器和数据库服务器的响应时间。对比 T_n 和 T_s，就很容易知道系统在哪些环节的响应时间比例较大。

只要判断出系统的瓶颈是出现在网络还是服务器端，就可以层层推进对相应环节的组件响应时间进行深入分析，直到找到造成性能问题的根本原因。

借助 LoadRunner 的分析组件 Analysis，很容易按照“由外而内，由表及里，层层深

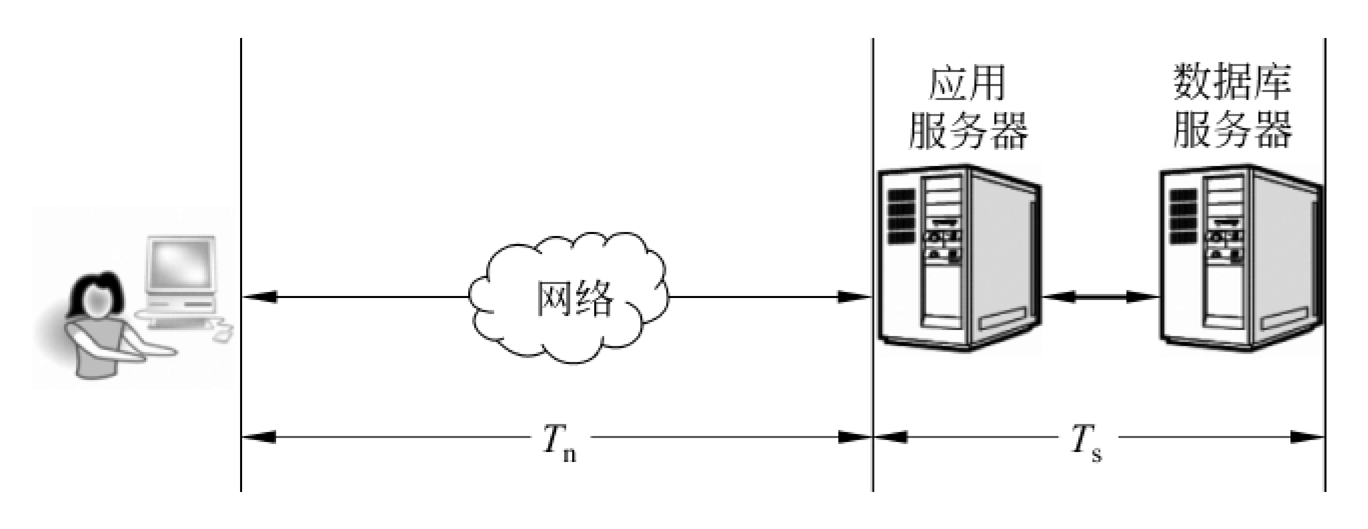


图 16-2　客户交易分解图

入”的原则进行分析，快速将问题定位。例如从图 16-3 中可以直接看出瓶颈出现在网络上。

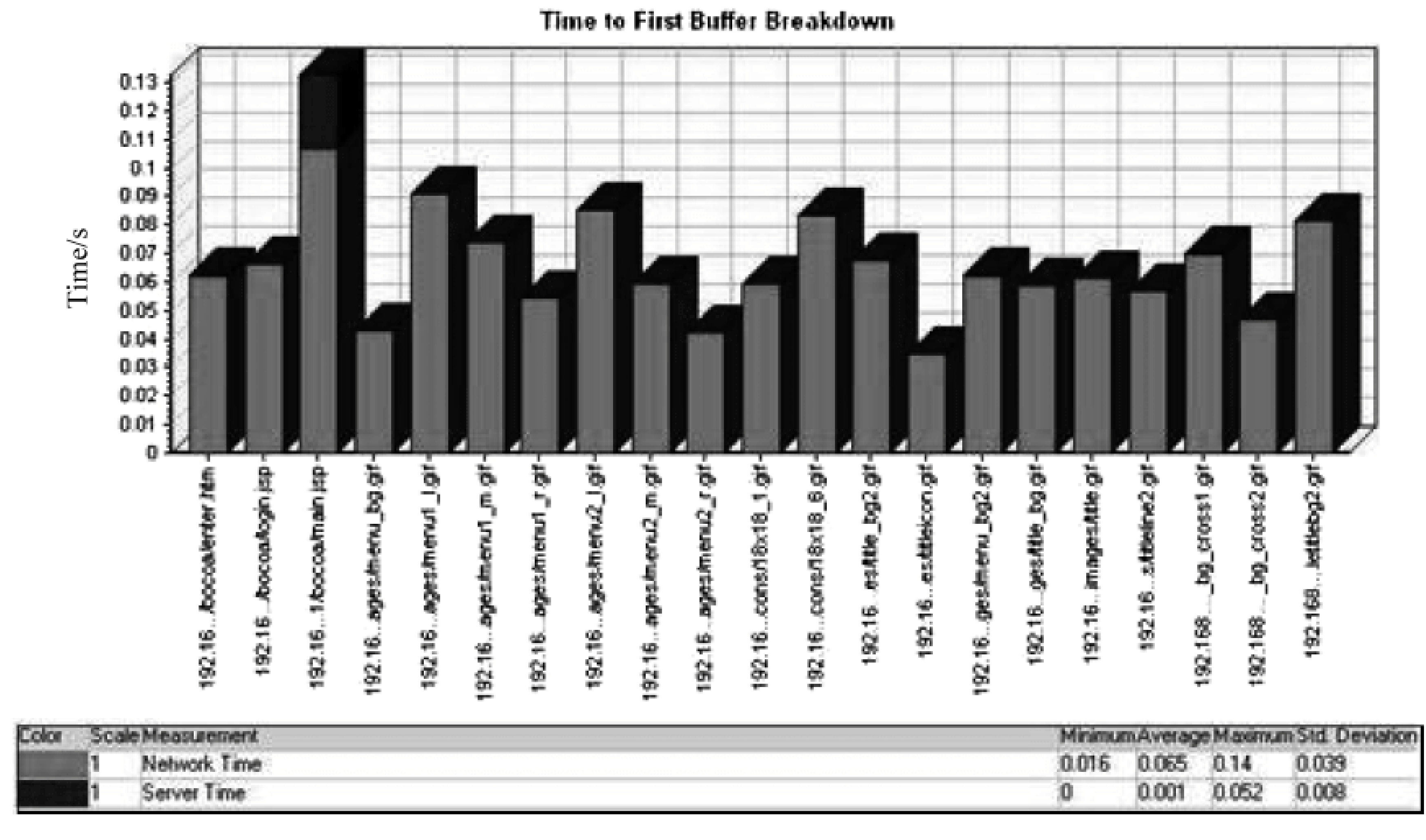


图 16-3　客户请求第一个 Buffer 的分解示例

看了前面的内容，也许很多人会以为性能分析非常容易，借助工具即可完成，但实则不然。即使有了正确的测试结果，也不一定能对系统的性能问题进行正确定位。例如，服务器的内存不够可能会引起较大的磁盘 I/O，进而导致 CPU 利用率居高不下，其根本原因可能是程序内部存在内存泄漏，而不是内存瓶颈。这类问题不但要靠经验，更要靠对系统的深入了解。

不难看出，性能测试是难度较大的一项工作，绝不是一蹴而就的事情。根据作者的经验，最好的办法是把性能分析贯穿于性能测试过程的始末，所有人员都应该给予高度关注。

实际上，性能测试分析从测试场景执行时就开始了，而不是仅仅在测试结束后才进行的。例如，在测试执行过程中可以借助分析数据库，观察事务实时响应时间来发现一些问题。

除了这些通用的方法外，性能测试分析人员还应该在测试设计、执行、分析等各阶段把工作做透，只有这样才能把性能测试工作做好。

16.2 挖掘 LR 中的错误信息

在测试场景执行过程中，LoadRunner 采集了虚拟用户、操作系统、应用服务器等各种运行数据，这些数据成为分析系统性能的重要参考资料。当测试场景运行结束后，就可以通过 Analysis 对这些测试结果进行专门的分析，以发现系统的潜在问题。

在测试结束并完成测试结果数据收集后，就可以启动 Analysis 打开测试结果文件，将其导入 Microsoft Access 数据库，然后按照设置的模板打开默认的结果分析图。通常的分析器默认界面如图 16-4 所示。

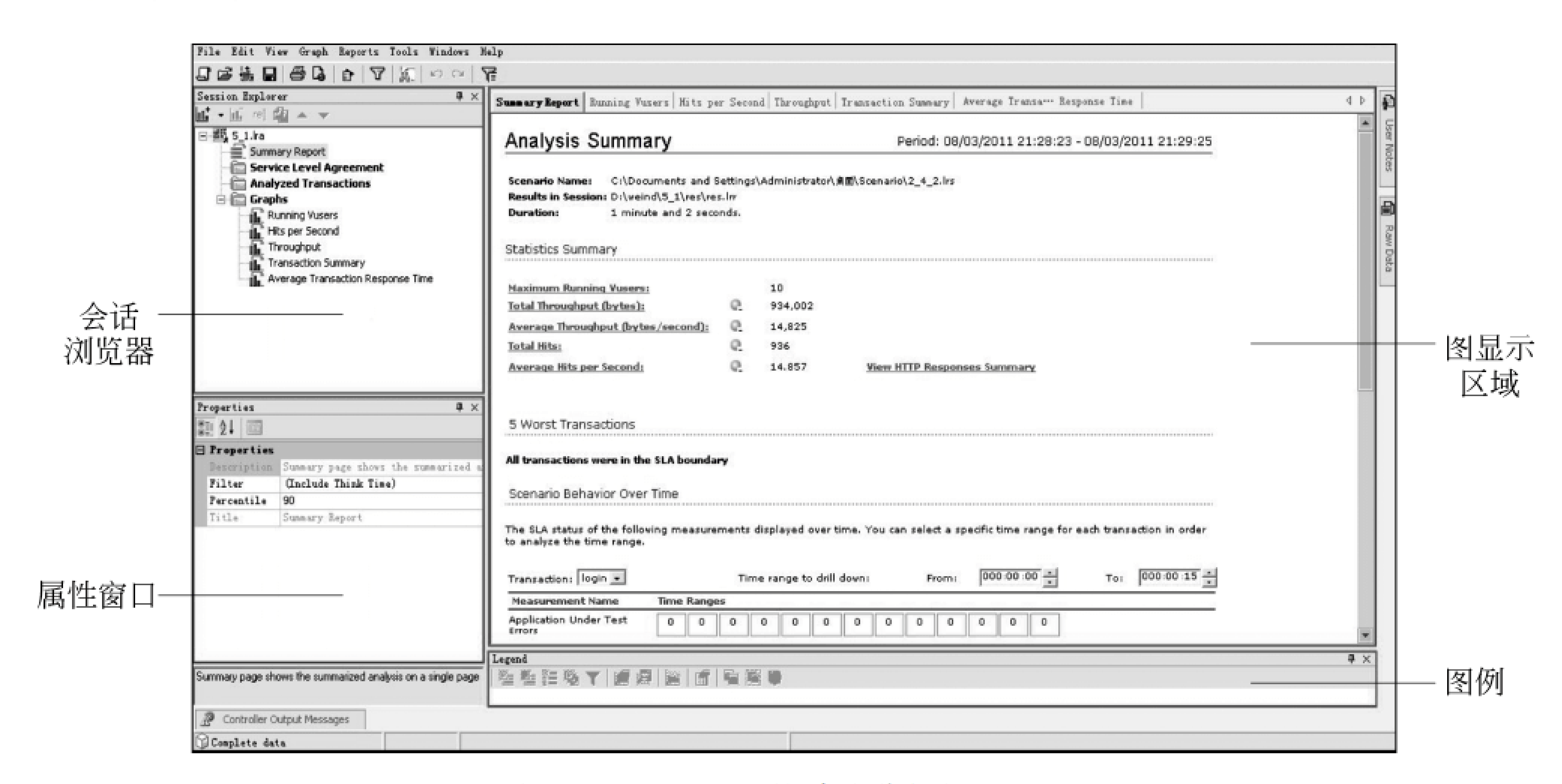


图 16-4　Analysis 的默认分析概要界面

利用 Analysis 进行分析的第一步是查看分析概要报告(Analysis Summary)，图 16-4 中显示的即为分析概要报告。分析概要报告展示了场景运行的统计信息、事务响应时间概述、HTTP 响应概述(对于 Web 测试)等。

在分析概要结果中，重点查看虚拟用户的运行情况(Statistics Summary)和事务综述(Transaction Summary)。对虚拟用户，主要查看最大并发用户数目；对事务综述，则要查看最大、最小、平均、90%事务最大响应时间、通过事务数量、失败事务数量等。

在场景运行时可以看到一些图，这些图将场景中的数据转化为折线图，方便人们了解当前该数据的状态。在默认情况下，Analysis 会自动打开如图 16-5 所示的几张分析图。可以通过菜单栏 Graphs 中的 Add New Graphs 命令完成添加图的操作，添加后弹出 Graphs 管理器，如图 16-6 所示。

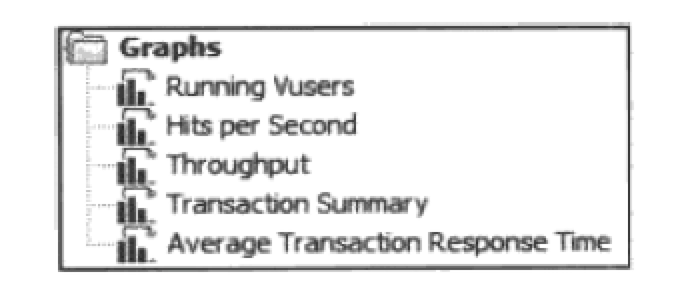


图 16-5　默认情况下系统打开的 Graphs

每张图都代表了场景运行中监控到的数据变化趋势，所以看懂每一张图的含义是性能分析的第一步，接着来介绍一些常见图的含义。

(1) 虚拟用户(Vusers)图：虚拟用户图分为运行状态的虚拟用户图、虚拟用户概要

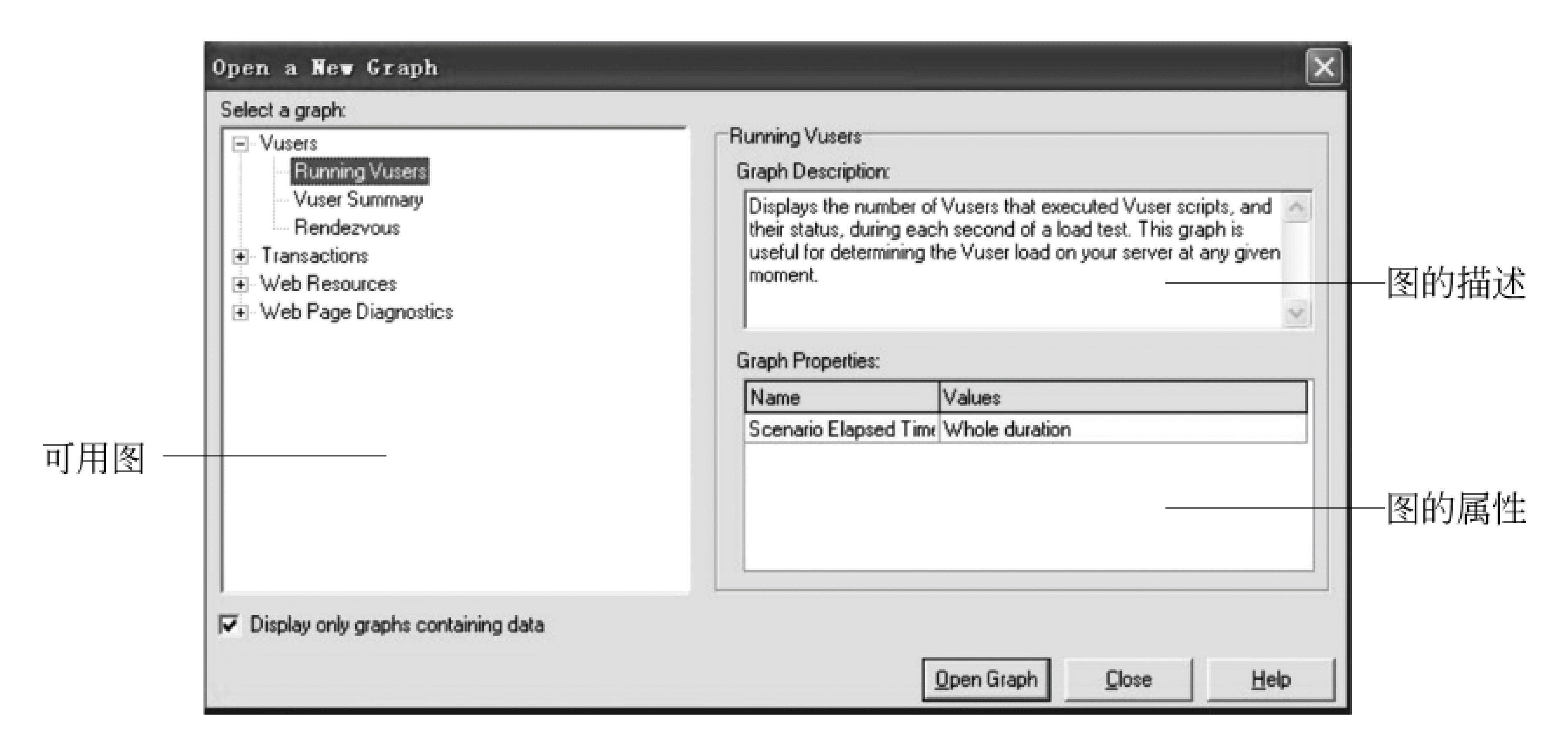


图 16-6 数据图管理界面

图和集合点图三类，主要借助其查看场景与会话的虚拟用户行为，可以帮助人们了解负载生成的过程。

① Running Vusers(负载过程中的虚拟用户运行情况)。

该图可以反映系统形成负载的过程，随着时间的推移，虚拟用户数是如何变化的。

在图 16-7 中可以看到用户在 9min 左右到达了负载峰值 50 个虚拟用户，负载的生成是大约每分钟增加 5 个用户，峰值负载持续 1min30s。

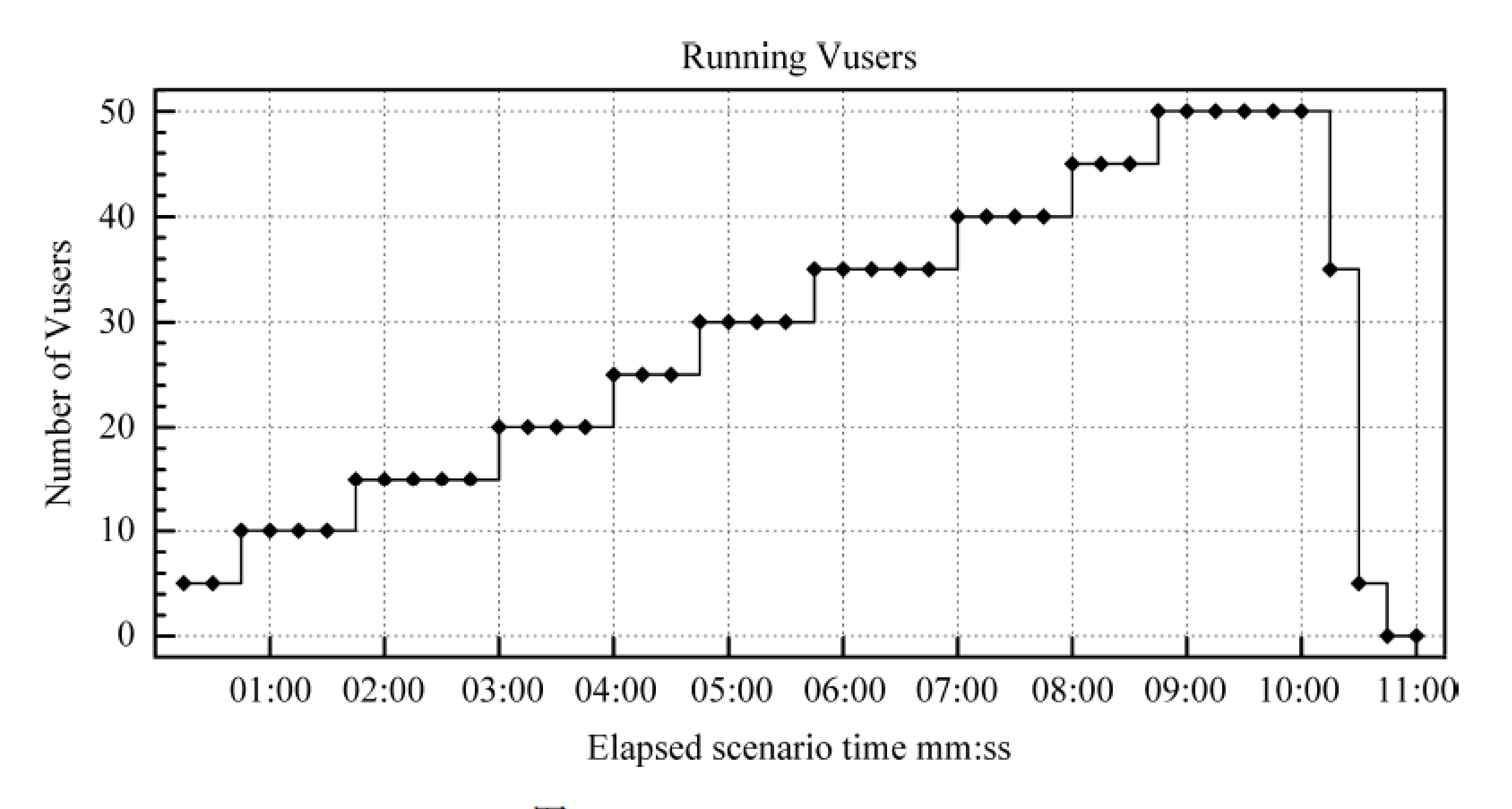


图 16-7 Running Vusers

② Rendezvous(负载过程中集合点下的虚拟用户数)。

当场景中设置了集合点后会出现这张图，该图反映了随着时间的推移各个时间点上并发用户的数目，方便人们了解并发用户数的变化情况。在图 16-8 中可以看到刚开始的 7min 内，负载的并发用户都是 1 个，而后面变化为 2 个用户并发。

(2) Errors 图：Errors 图主要有错误统计、每秒错误数量两类。借助 Errors 图可以发现服务器什么时间发生错误以及错误的统计信息，帮助人们定位产生错误的原因，可以分析服务器的处理能力。

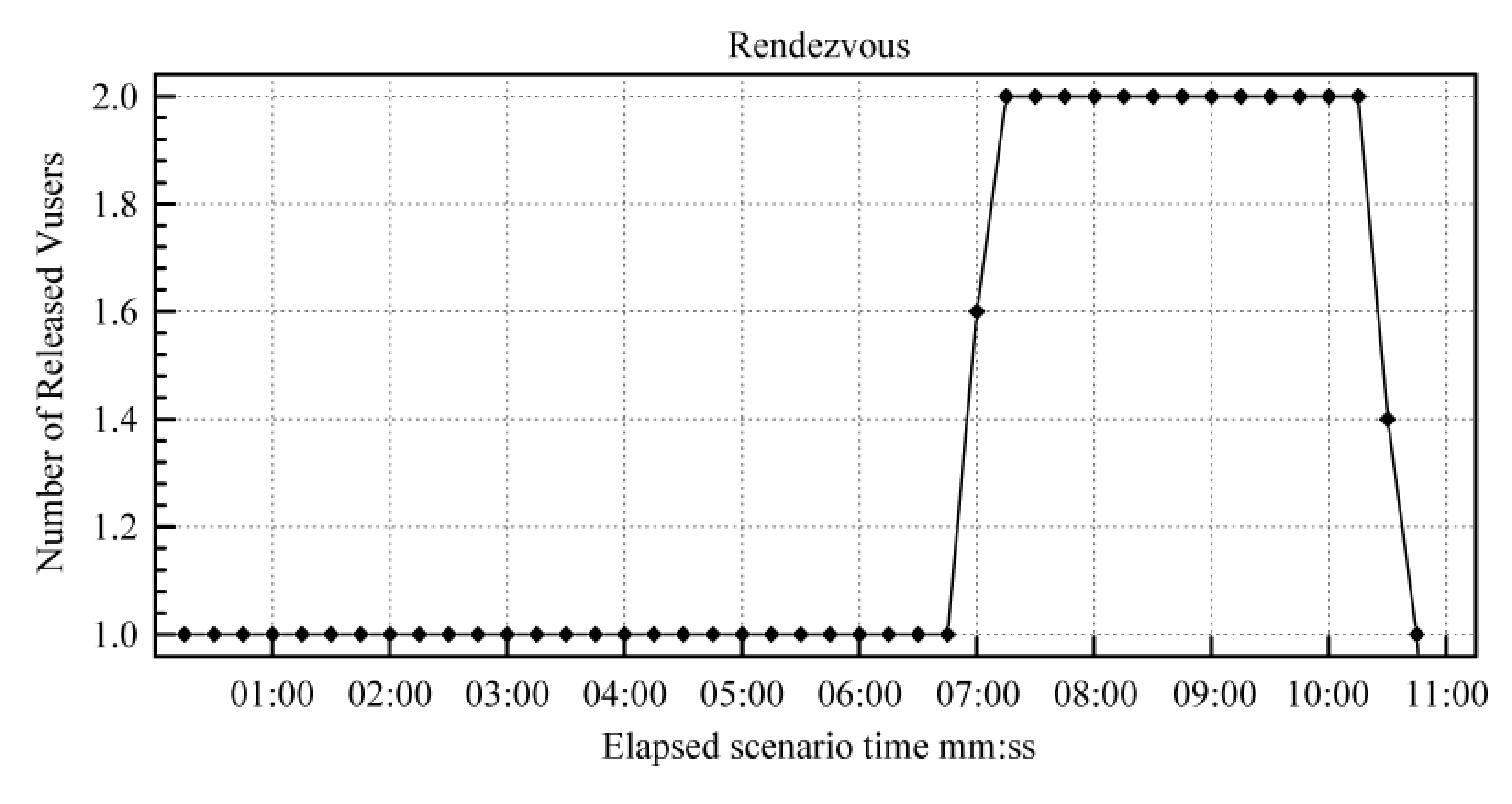


图 16-8 Rendezvous

通过每秒错误数可以了解在每个时间点上错误产生的数目，该数据越小越好。通过这个图可以了解错误随负载的变化情况，定位何时系统在负载下开始不稳定甚至出错，配合系统日志可以定位产生错误的原因。

在图 16-9 中可以看到场景在 37s 的时候出现了一次错误。

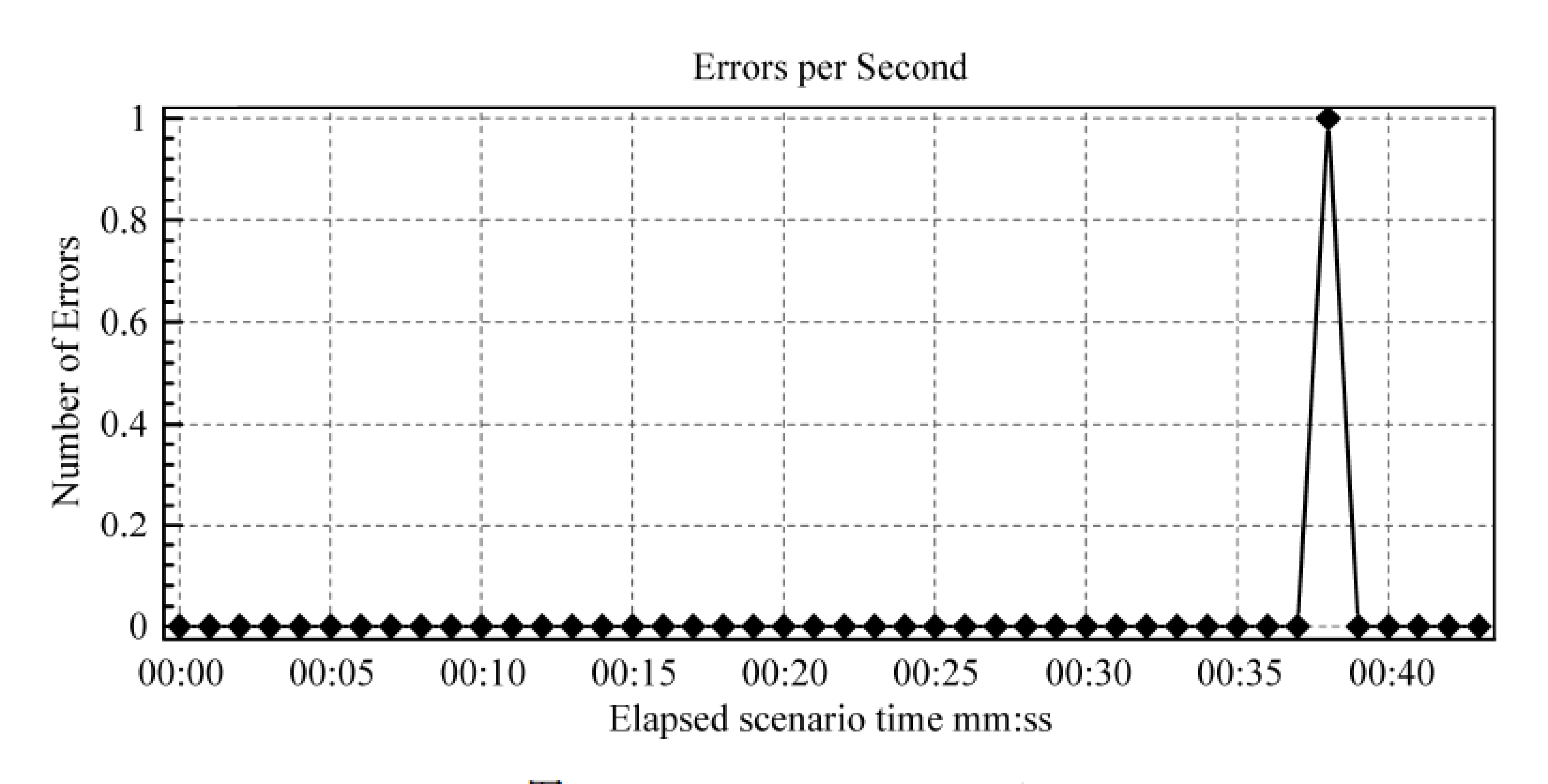


图 16-9 Errors per Second

(3) 事务(Transactions)图：Analysis 和事务相关的分析图表有事务综述图、事务平均响应时间图、每秒通过事务数图、每秒通过事务总数图、事务性能摘要图、事务响应时间与负载分析图、事务响应时间(百分比)图、事务响应时间分布图等，给出了所有和事务相关的数据统计，通过这些图表可以很容易分析应用系统事务的执行情况。

① Average Transaction Response Time(平均事务响应时间)。

这是人们比较关心的数据之一，反映随着时间的变化事务响应时间的变化情况，时间越小说明处理的速度越快。如果和前面的用户负载生成图合并在一起看，就可以发现用户负载增加对系统事务响应时间的影响规律。

在图 16-10 中可以看到响应时间是如何增长的，随着时间的推移响应时间逐渐变长，并且在不到 8min 的时候突然出现响应时间大幅下降的情况。

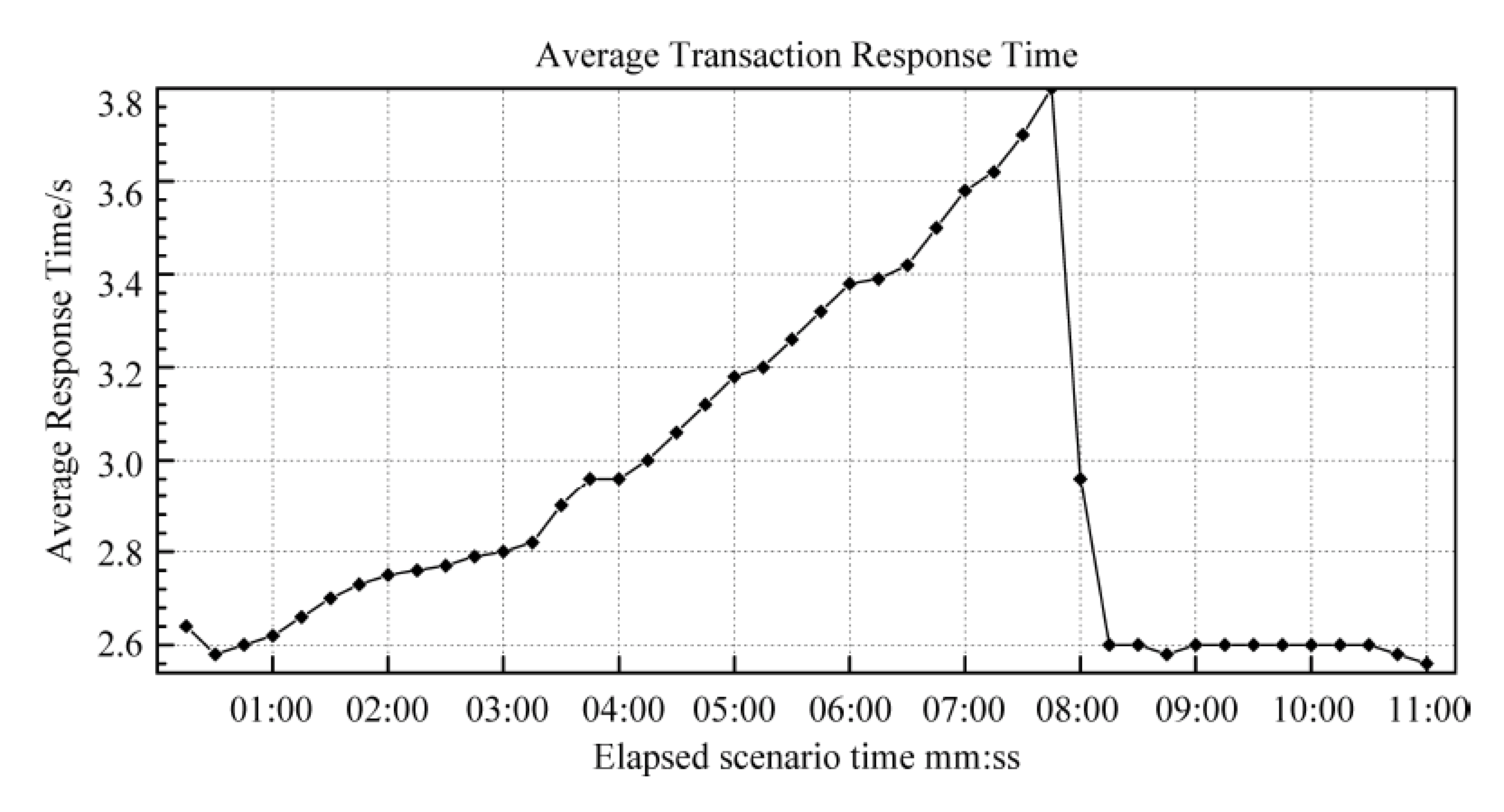


图 16-10　Average Transaction Response Time

另外事务的响应时间也不应该超过用户的最大接受范围，否则会出现系统响应过慢的问题。

② Transactions per Second（每秒事务数）。

另一个关键数据是 TPS 吞吐量，该数据反映了系统在同一时间内能处理业务的最大能力，这个数据越高，说明系统处理能力越强。

在图 16-11 中上面的线是通过的事务，下面的线是失败的事务，这里可以看到系统的 TPS 随着时间的变化逐渐变大，而在不到 10min 的时候系统每秒可以处理 1.9 个事务。这里的最高值并不一定代表系统的最大处理能力，TPS 会受到负载的影响，也会随着负载的增加而逐渐增加，当系统进入繁忙期后，TPS 会有所下降，而在 4min 以后开始出现少量的失败事务。

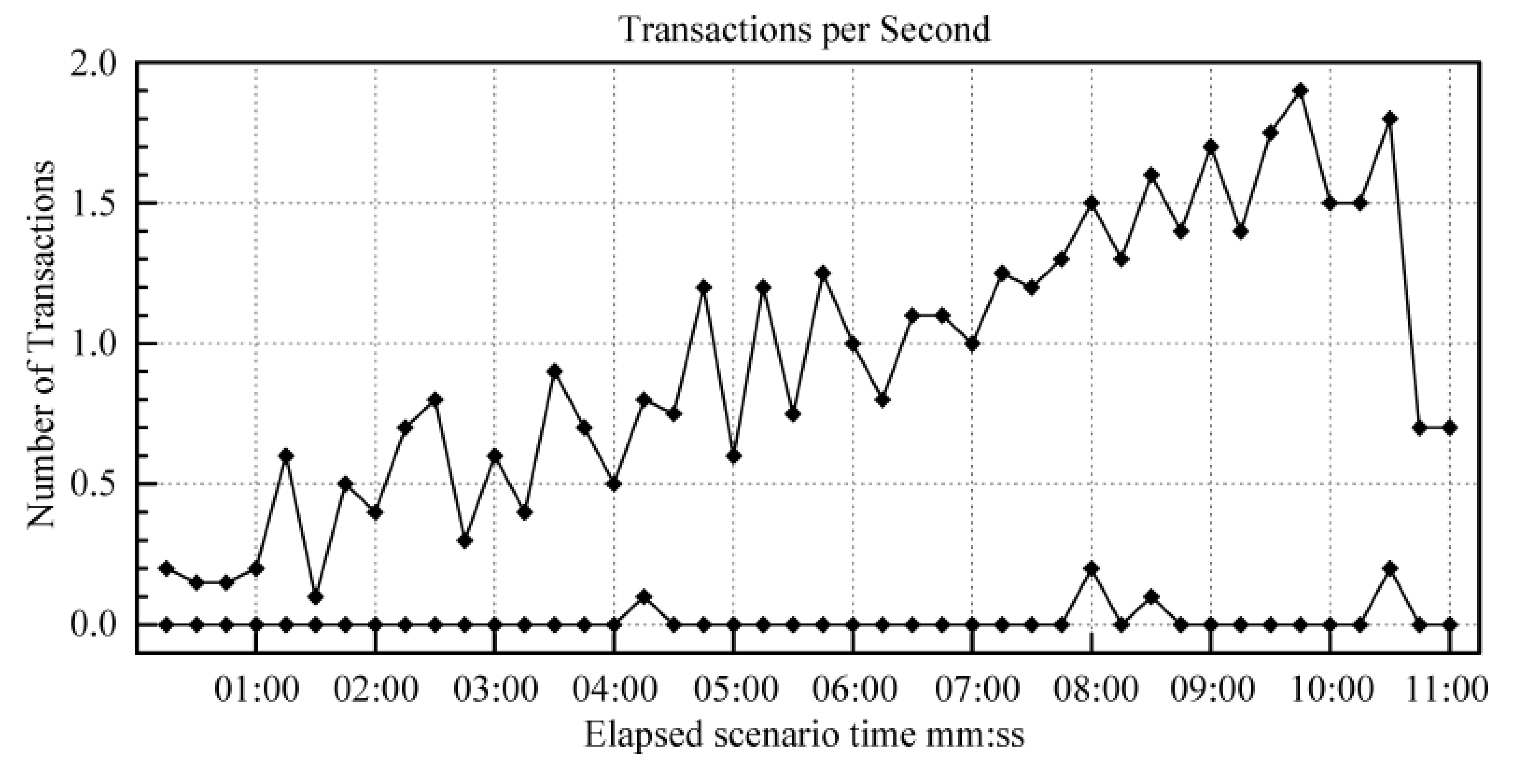


图 16-11　Transactions per Second

③ Transaction Summary（事务概要说明）。

该说明给出事务的 Pass 个数和 Fail 个数，了解负载的事务完成情况。通过的事务数越多，说明系统的处理能力越强；失败的事务越少，说明系统越可靠。

在图16-12中可以看出，对于reg注册操作一共有613次操作成功，有6次失败。可以考虑结合前面的每秒错误数进一步分析为什么会出现6个注册错误，以及错误发生的时间和该时间产生错误的原因。

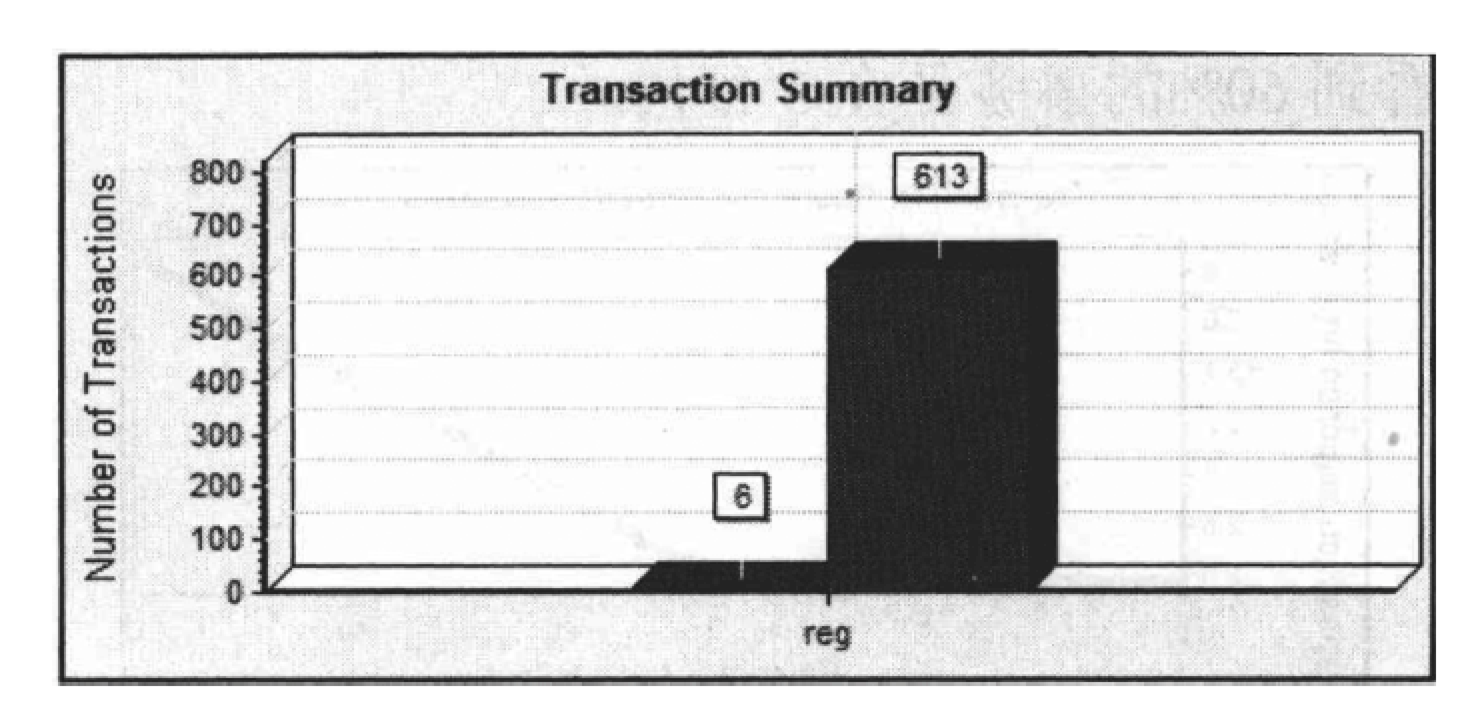


图16-12　Transaction Summary

④ Transaction Performance Summary(事务性能概要)。

这里会给出事务的平均时间、最大时间、最小时间柱状图，方便分析事务响应时间的情况。在图16-13中可以看到reg这个事务的最大时间为3.897s，最小时间为2.555s，平均时间为2.924s。柱状图的落差越小说明响应时间的波动就越小，如果落差很大，那么说明系统不够稳定。

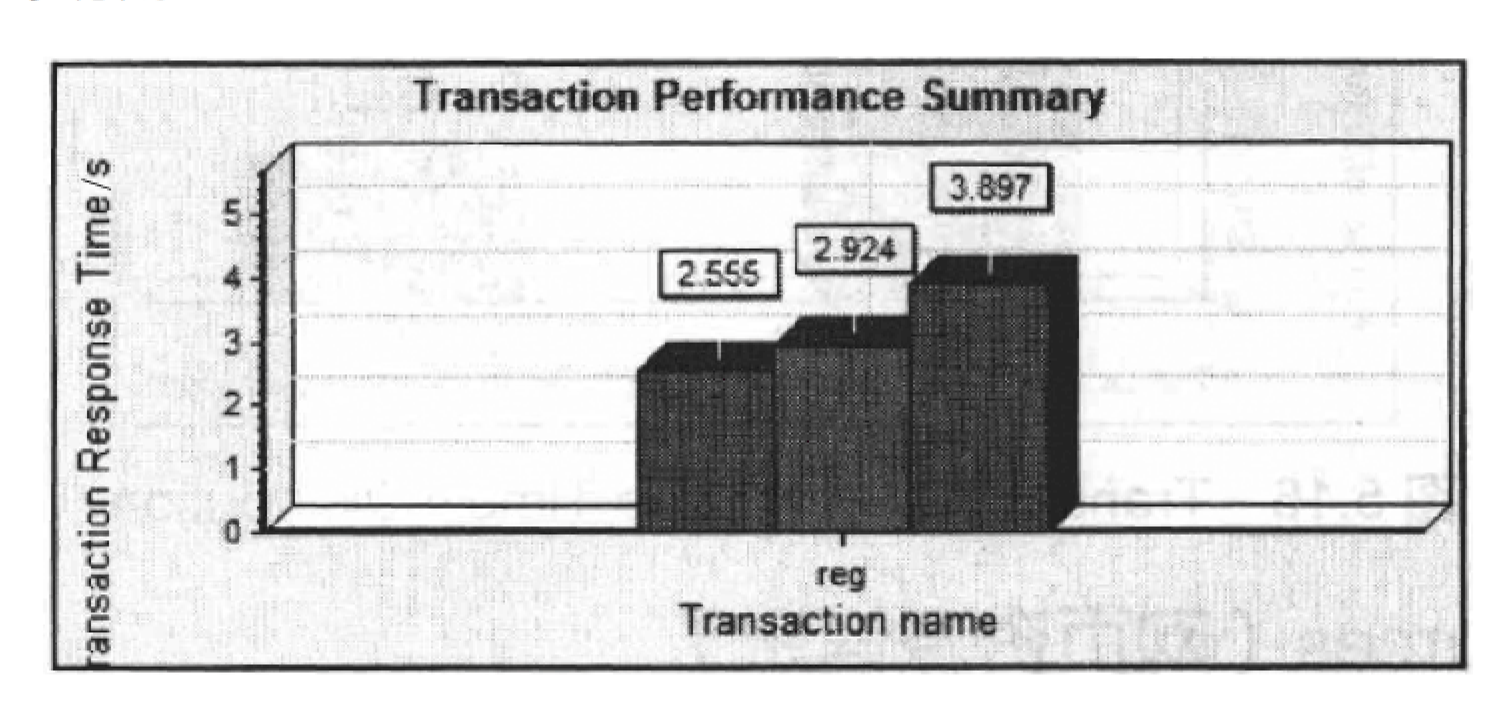


图16-13　Transaction Performance Summary

⑤ Transaction Response Time Under Load(在用户负载下事务响应时间)。

这里给出了在负载用户增长的过程中响应时间的变化情况，其实这张图也是将Vusers和Average Transaction Response Time图做了一个Correlate Merge得到的，该图的线条越平稳，说明系统越稳定。在图16-14中可以看出在负载逐渐增加到5个用户时，事务的响应时间基本没有变化。而用户增加到15个开始，随着用户负载的增加响应时间也有较大的波动。

⑥ Transaction Response Time(Percentile)(事务响应时间的百分比)。

这里给出的是不同百分比下的事务响应时间范围，通过这张图可以了解有多少比例的事务发生在某个时间内，也可以发现响应时间的分布规律，数据越平稳说明响应时间变化越小。在图16-15中可以看到60%的事务是在3s内。

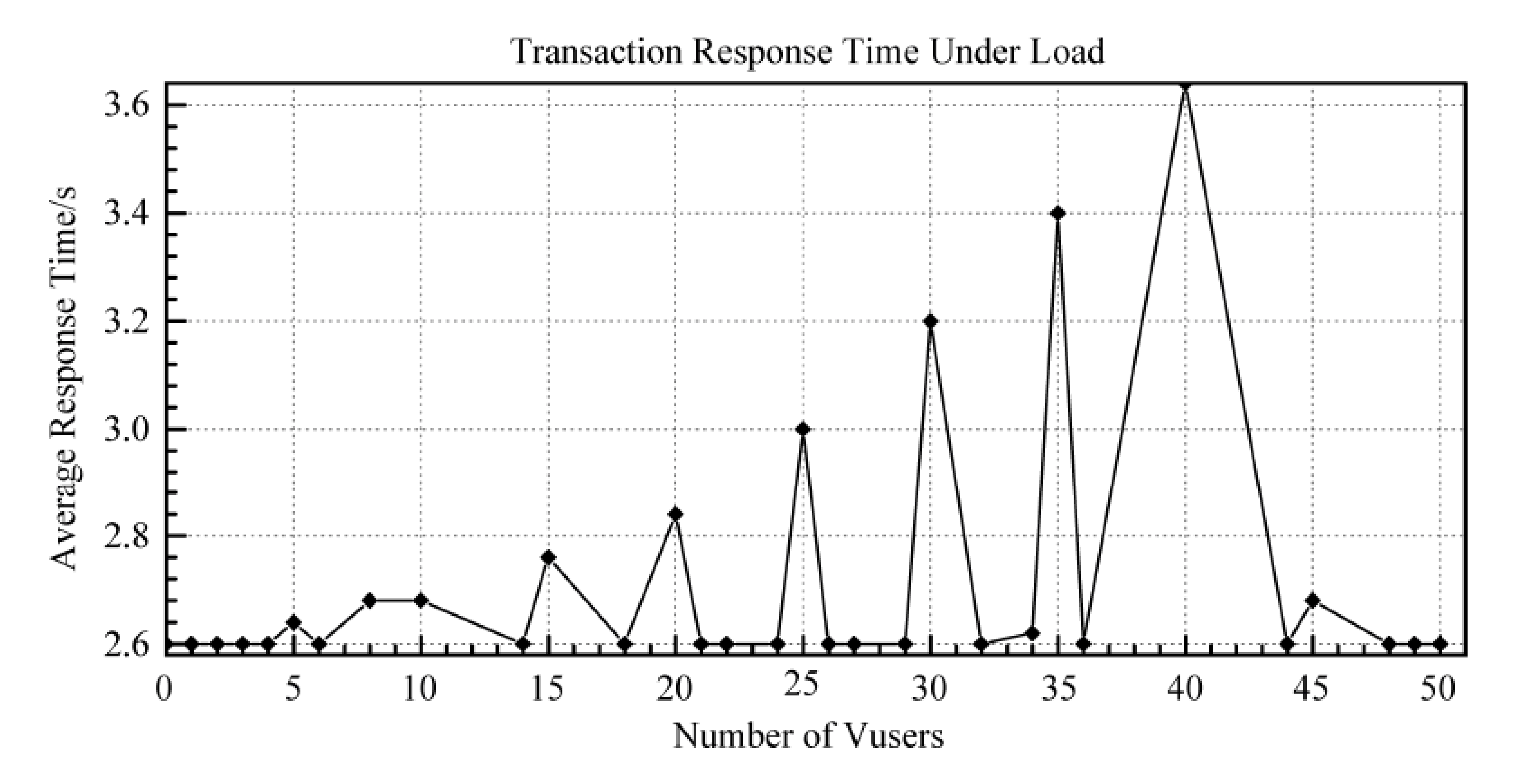


图 16-14　Transaction Response Time Under Load

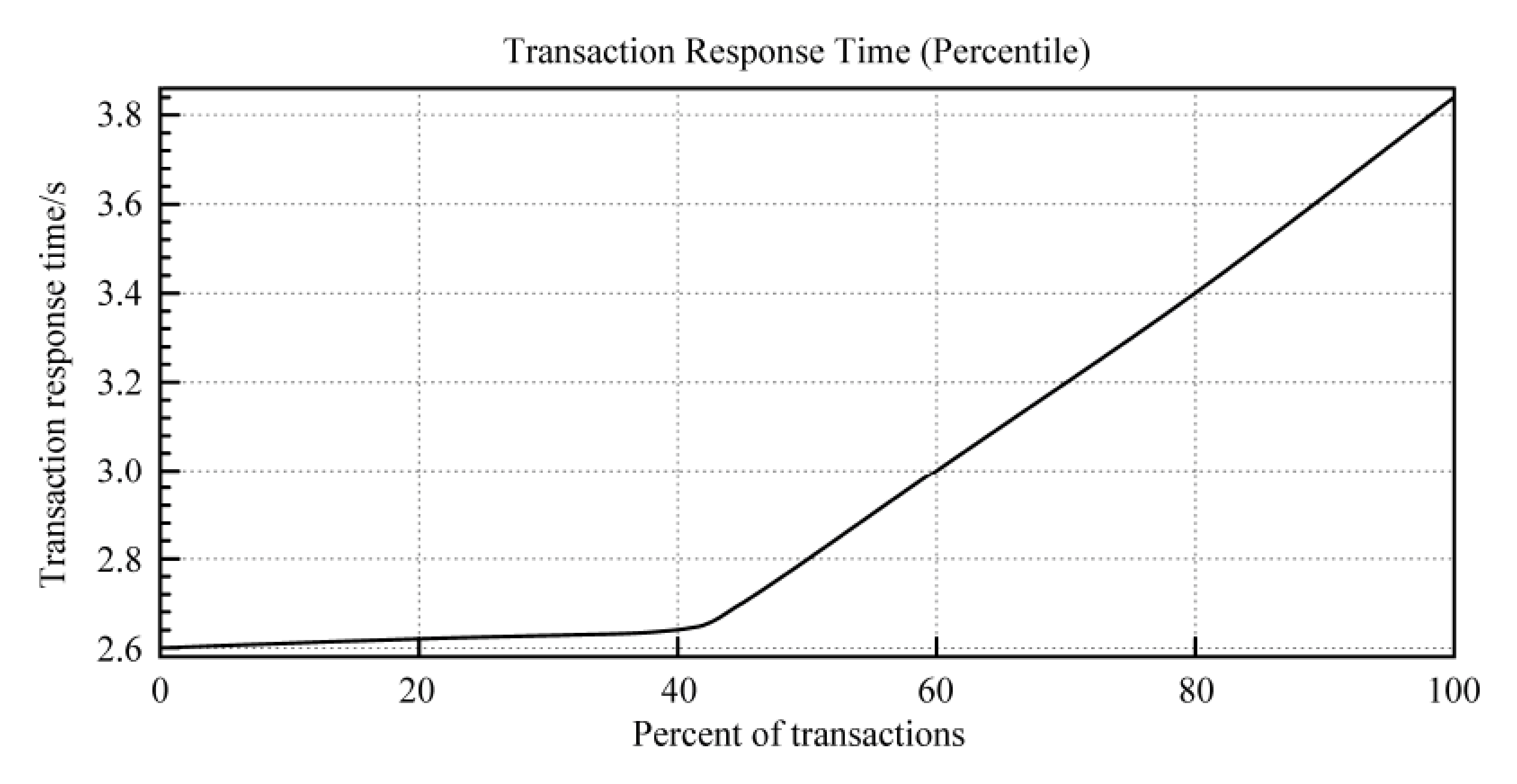


图 16-15　Transaction Response Time(Percentile)

⑦ Transaction Response Time(Distribution)(每个时间段上的事务数)。

图 16-15 给出的是在每个时间段上的事务个数,响应时间较小的分类下的事务数越多越好。从图 16-16 中可以看到在所有的事务中,有 391 个事务的响应时间最接近 2s,而有 222 个事务的响应时间最接近 3s。

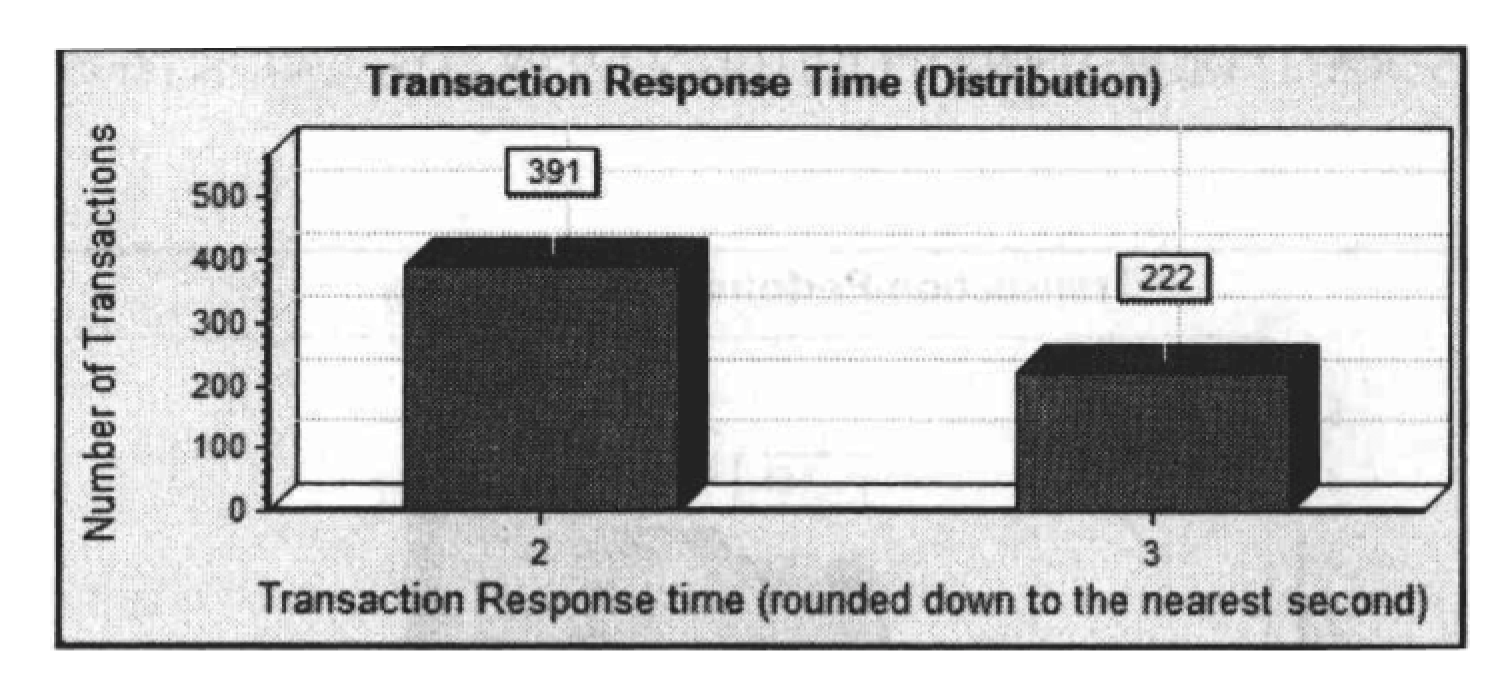


图 16-16　Transaction Response Time(Distribution)

(4) Web资源(Web Resources)图：Web资源图主要有Web服务器的吞吐率图、点击率图、返回的HTTP状态代码图、每秒HTTP响应数图、每秒重试次数图、重试概述图、服务器连接数概要图、服务器每秒建立的连接数量图等。给出的是对于Web操作的一些基本信息，借助Web资源图，可以深入地分析服务器的性能。

① Hits per Second(每秒单击数)。

每秒单击数提供了当前负载中对系统所产生的单击量记录。每一次单击相当于对服务器发出了一次请求，一般单击数会随着负载的增加而增加，该数据越大越好。

在图16-17中可以看出随着时间的增加，每秒单击数在上升，最高达到78次/s。

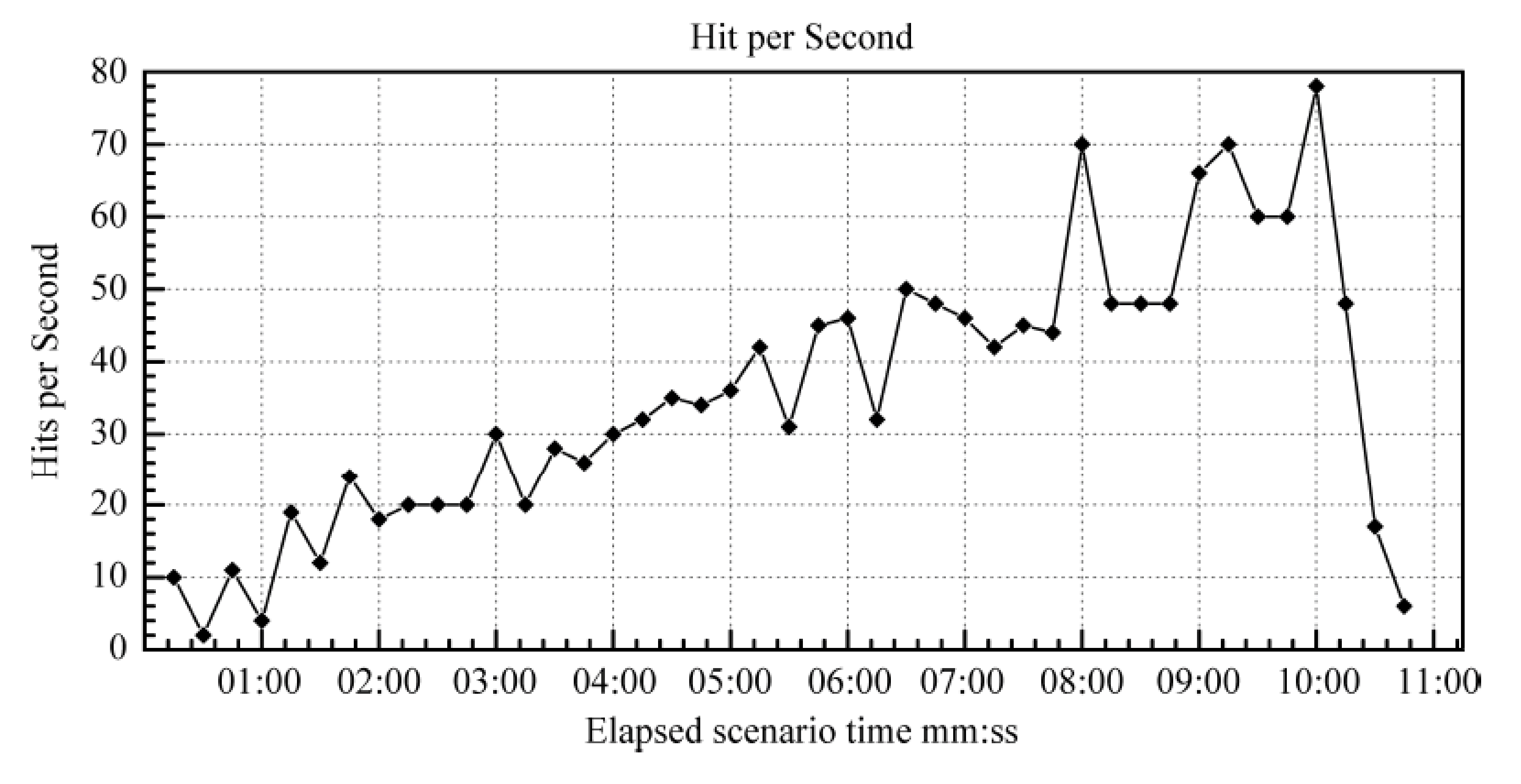


图16-17 Hits per Second

② Throughput(带宽使用)。

这里给出了在当前系统负载下所使用的带宽，该数据越小说明系统的带宽依赖越小，通过这个数据能确定是否出现了网络带宽的瓶颈(注意这里使用的单位是字节)。

在图16-18中可以得到最高的带宽峰值是355 000B，远远低于100Mb的局域网带宽上限，所以系统不存在带宽瓶颈。

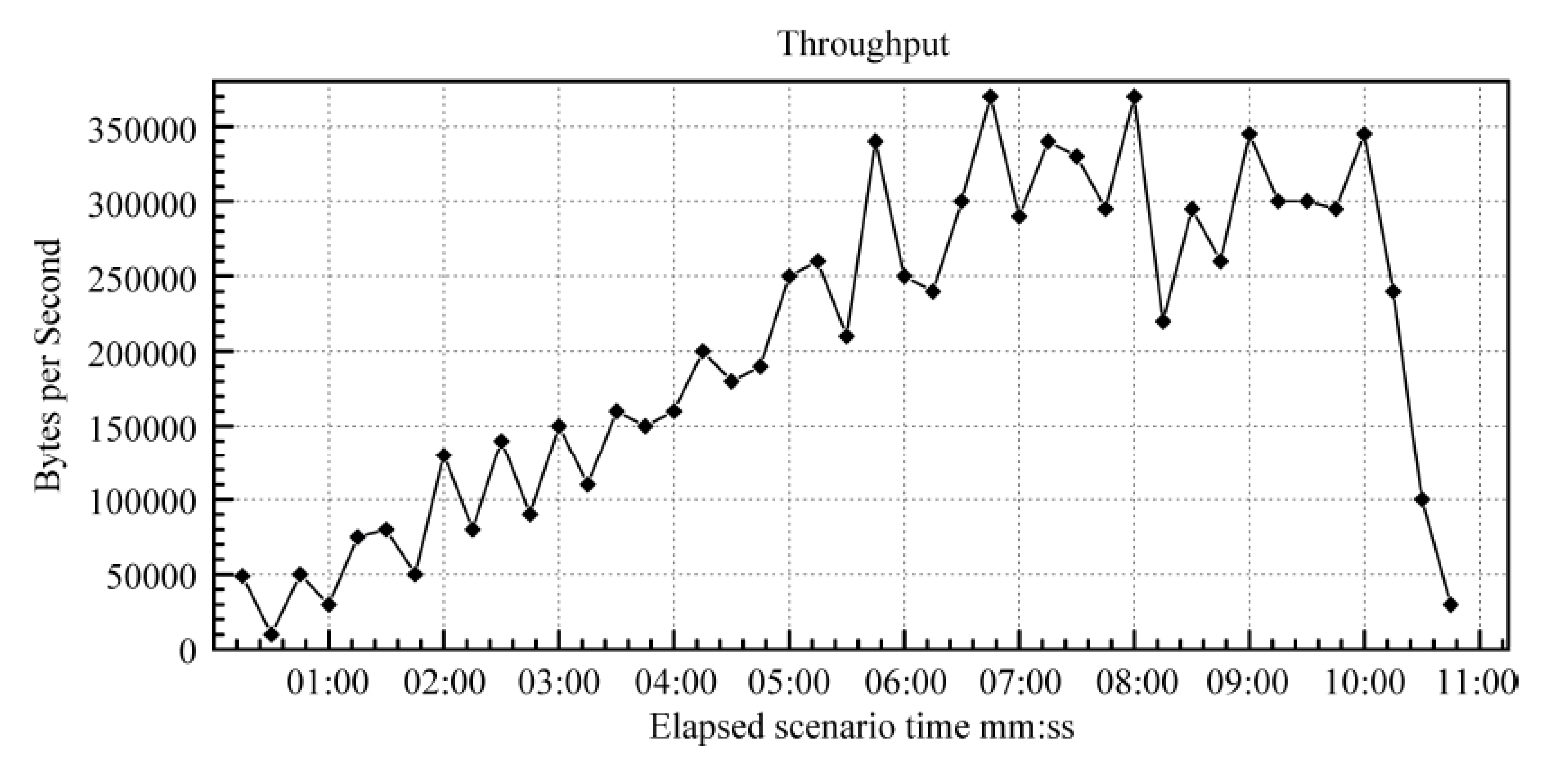


图16-18 Throughput

③ HTTP Responses per Second(每秒 HTTP 响应数)。

这里给出了每秒钟服务器返回各种状态的数目，该数值一般和每秒单击量相同。单击量是指客户端发出的请求数，而 HTTP 响应数是指服务器返回的响应数。如果服务器返回的响应数小于客户端发出的单击数，那么说明服务器无法应答超出负载的连接请求。在图 16-19 中可以看到最高峰时服务器每秒能返回接近 75 个 HTTP 200 OK 的状态。

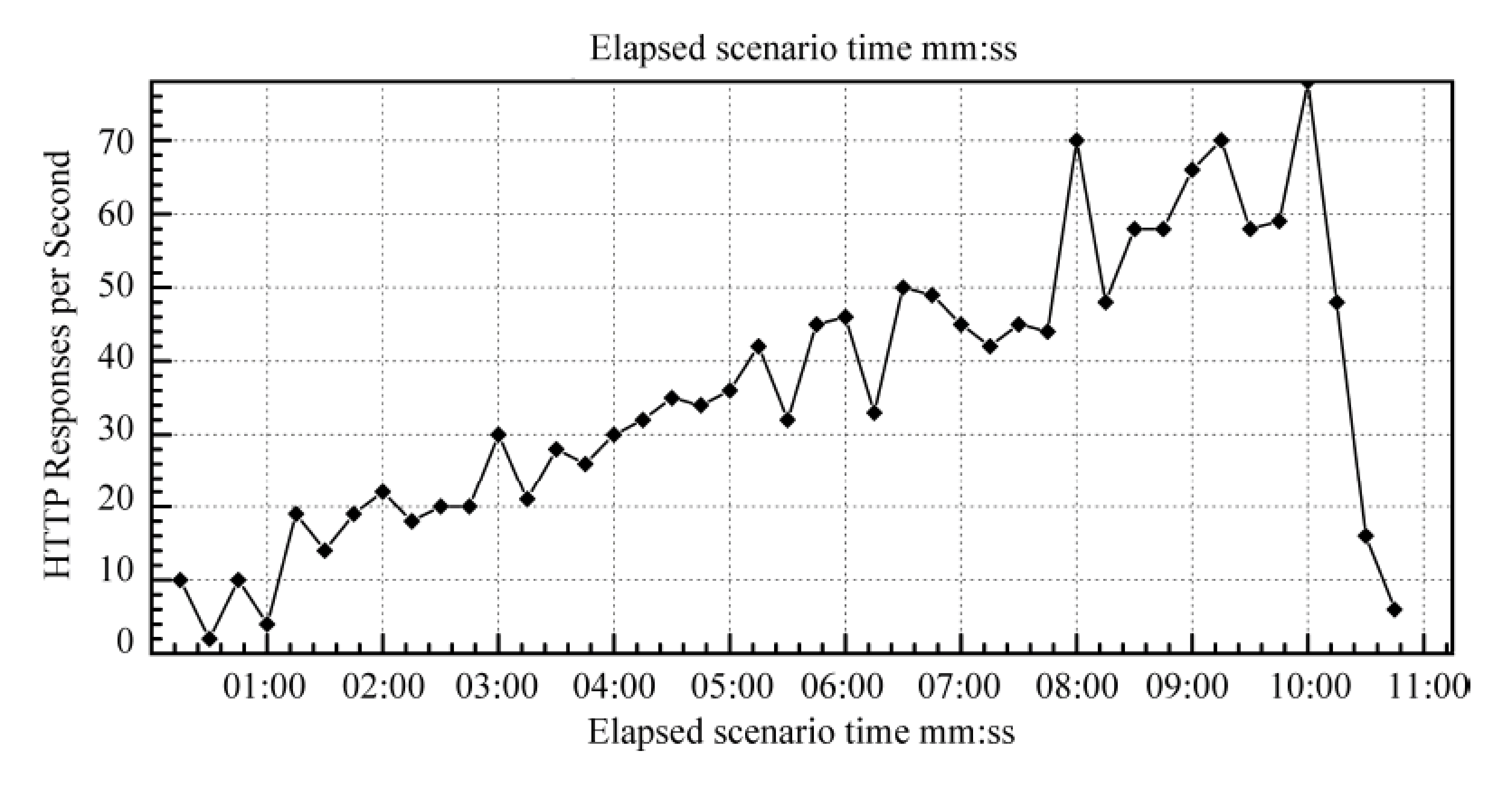


图 16-19 HTTP Responses per Second

这个数据和前面的每秒单击数吻合，说明服务器能够对每一个客户端请求进行应答。

④ Connections Per Second(每秒连接数)。

这里会给出两种不同状态的连接数，即中断的连接和新建的连接，方便用户了解当前每秒对服务器产生连接的数量。

在图 16-20 中可以看到随着时间的推移，系统的连接数逐步上升，最高达到每秒 4 个连接。

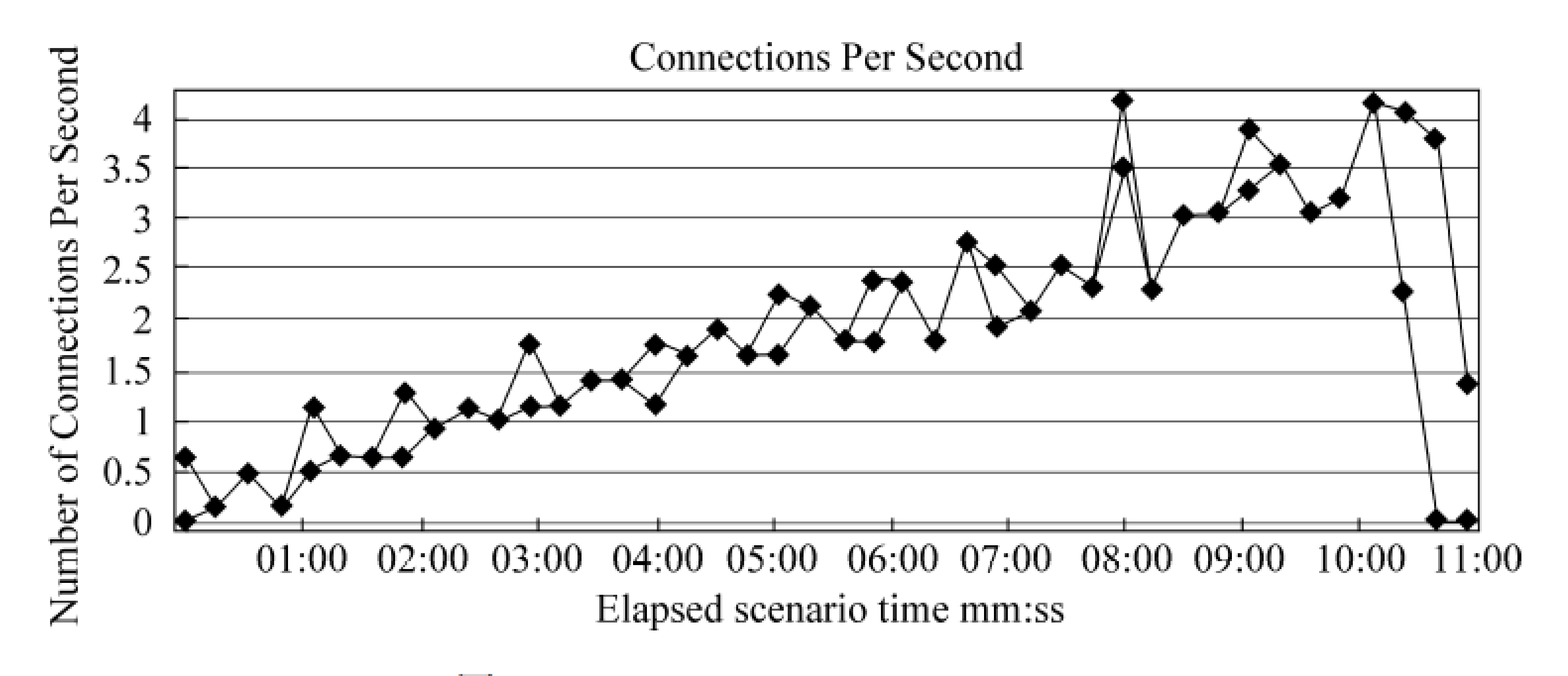


图 16-20 Connections Per Second

同时的连接数越多，说明服务器的连接池越大，当连接数随着负载上升而停止上升时，说明系统的连接池已满，无法连接更多的用户，通常这个时候服务器会返回 504 错误。可以通过修改服务器的最大连接数来解决该问题。

(5) 网页细分(Web Page Breakdown)图：在 Controller 中启动网页细分功能后，才可以在 Analysis 中查看网页细分图，启动细分功能的具体步骤是：在 Controller 菜单中

选择 Diagnostics→Distribution 进入如图 16-6 所示的界面，在图 16-21 中同时选中 Enable the following diagnostics 和 Web Page Diagnostics(Max Allowed Distribution 10%)复选框。

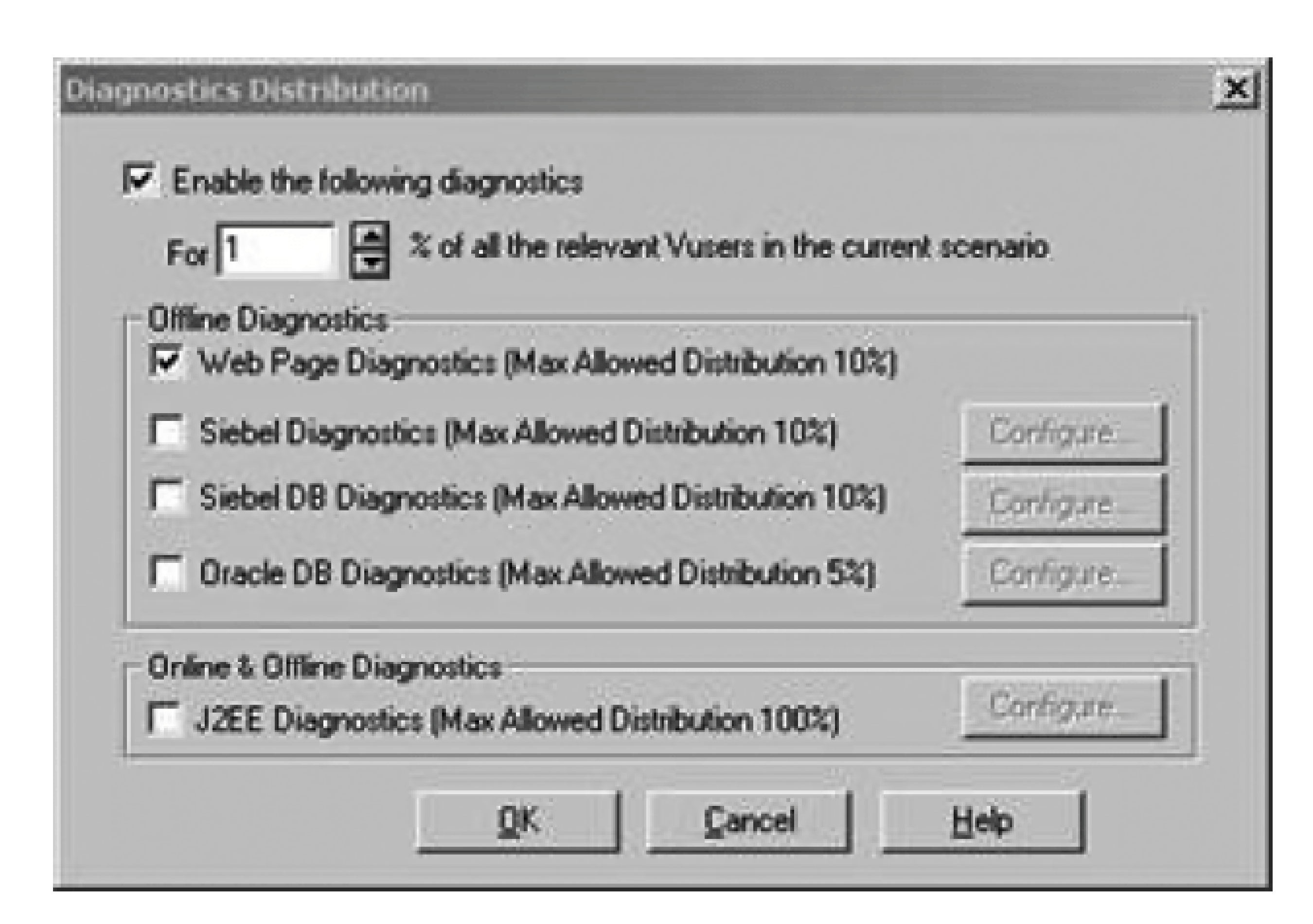


图 16-21 启动网页细分图功能

网页细分图主要有页面分解总图、页面组件细分图、页面组件分解(随时间变化)图、页面下载时间细分图、页面下载时间细分(随时间变化)图、第一次缓冲时间细分图、第一次缓冲时间细分(随时间变化)图、已下载组件大小图。通过这个图，可以对事务的组成进行抽丝剥茧的分析，得到组成这个页面的每一个请求时间分析，进一步了解响应时间中有关网络和服务器处理时间的分配关系。通过这个功能，可以实现对网站的前端性能分析，明确系统响应时间较长是由服务器端(后端)处理能力不足还是客户端连接到服务器的网络(前端)消耗导致的。

① Web Page Diagnostics(网页分析)。

添加该图先会得到整个场景运行后虚拟用户访问的 Page 列表，也就是所有页面下载时间列表。这里对 Discuz. Net 论坛的注册用户事务进行分析。在图 16-22 中可以看到整个负载由三个页面请求组成，其中有一个请求始终在 0.8s 以内，而另外两个请求时间较长并且有上升趋势。

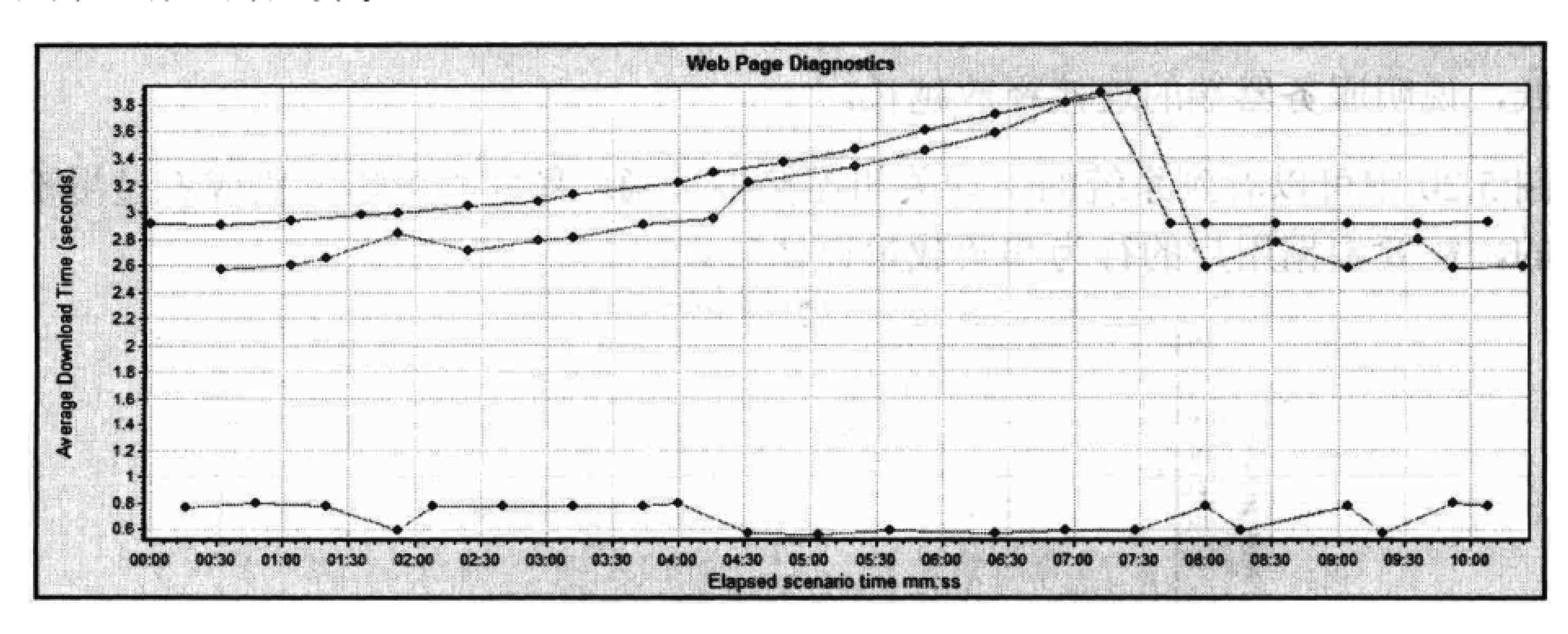


图 16-22 Web Page Diagnostics

然后通过 Select Page to Break Down 命令选择具体的 Page 来获得每个请求的相关详细信息。这里选择创建用户的 172.168…x? createuser=1 请求进行分析。稍后可以在图 16-23 中看到创建用户响应时间的变化,随着时间的增长响应时间从 2.6s 上升到了 3.9s,并且在 7min30s 时大幅下滑,回到 2.6s 左右。

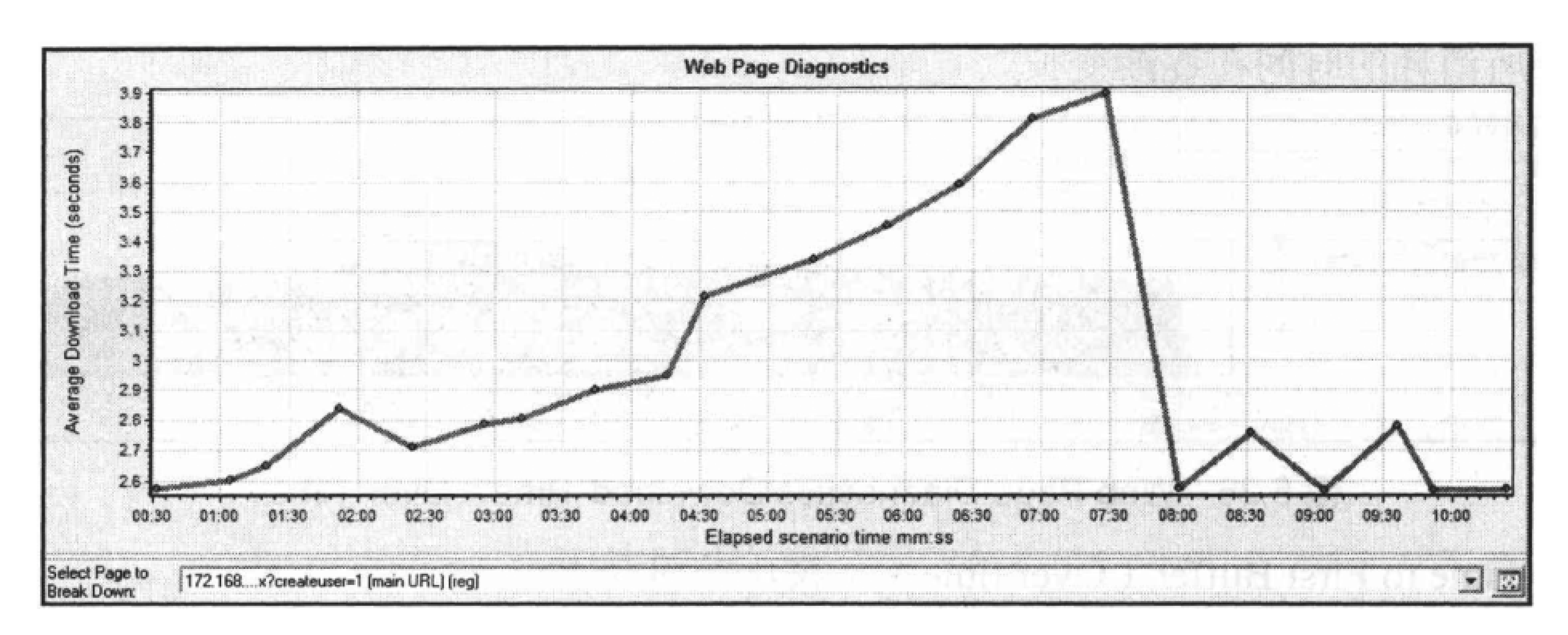


图 16-23 Web Page Diagnostics 对用户注册页面细分

在 Diagnostics options 选项中提供了以下 4 大块功能。

a. Download Time(下载时间分析)。

这里可以得到组成页面的每个请求下载时间。在图 16-24 中可以看到创建用户的操作由 4 个请求组成,其中导致注册用户较慢的主要原因是注册完成后需要等待两秒钟再刷新至论坛首页,而非注册用户本身需要消耗时间。首页刷新慢也只是因为 Client(客户端)需要消耗较多的时间,同时 Receive(接收)的时间也有一定的影响。

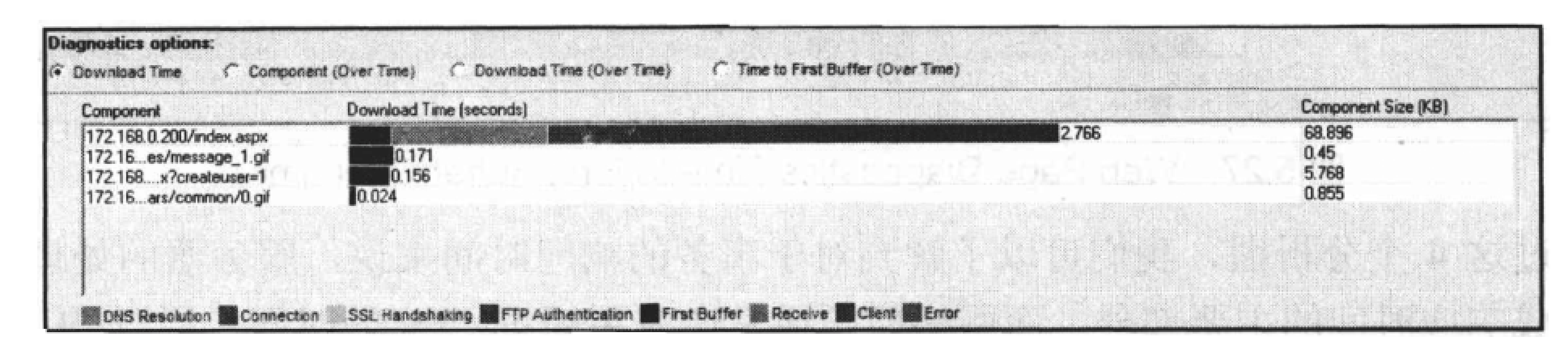


图 16-24 Web Page Diagnostics Download Time

b. Component(Over time)(各模块的时间变化)。

这里列出组成页面的每个元素,以及随着时间的变化所带来的响应时间变化。通过这个功能可以分析响应时间变长是因为页面生成慢,还是因为图片资源下载慢。在图 16-25 中可以发现随着时间的增加,首页的处理时间(最上面的一根线)从 0.5s 上升到了最大值 1.6s,而注册用户响应时间几乎没有上升。

c. Download Time(Over time)(模块下载时间)。

这里提供了针对每个组成页面元素的时间组成部分分析,方便确认该元素的处理时间组成部分。在图 16-26 中可以发现首页请求的下载时间主要消耗在 Client 上,而 7min30s 之前 Recevie 所消耗的时间在逐渐变长。

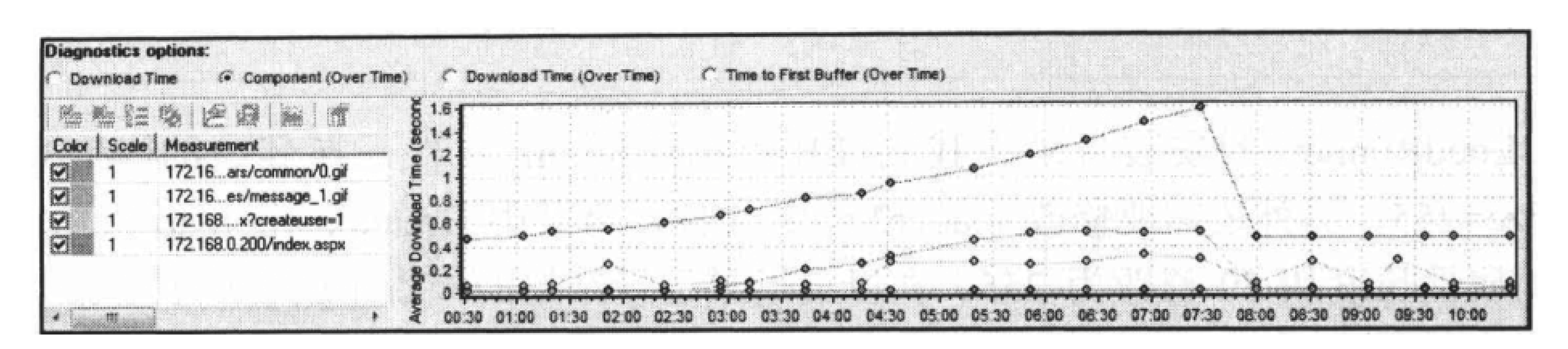


图 16-25　Web Page Diagnostics Component(Over Time)

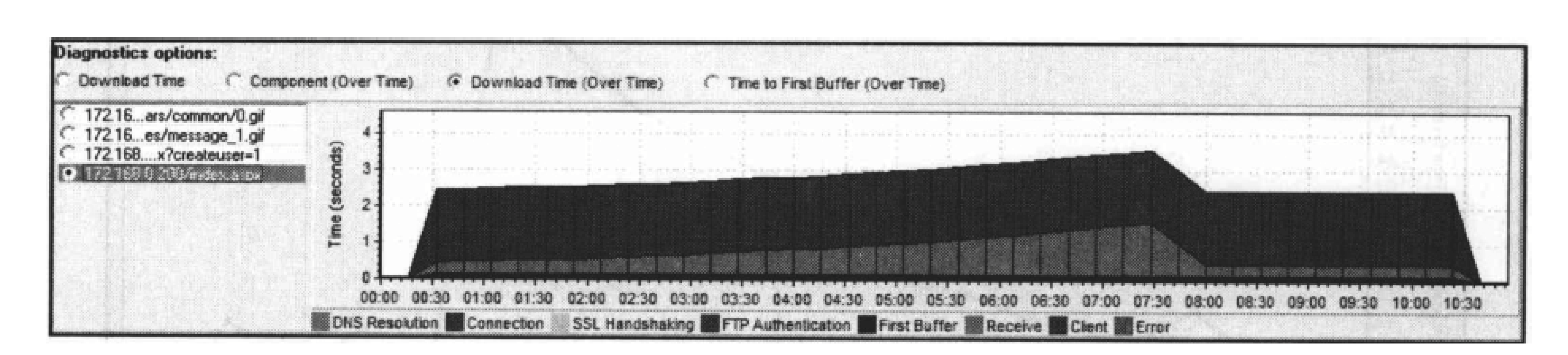


图 16-26　Web Page Diagnostics Download Time(Over Time)

d. Time to First Buffer(Over time)(模块时间分类)。

这里会列出该元素所使用的时间分配比例，是受 Network Time 影响多还是 ServerTime 影响多。在图 16-27 中可以看出，对于首页刷新的响应时间来说，主要是 Network Time 网络上消耗的时间，而 Server Time 服务器端的处理是非常优秀的。Server Time 是指服务器对该页面的处理时间；Network Time 是指网络上的时间开销。

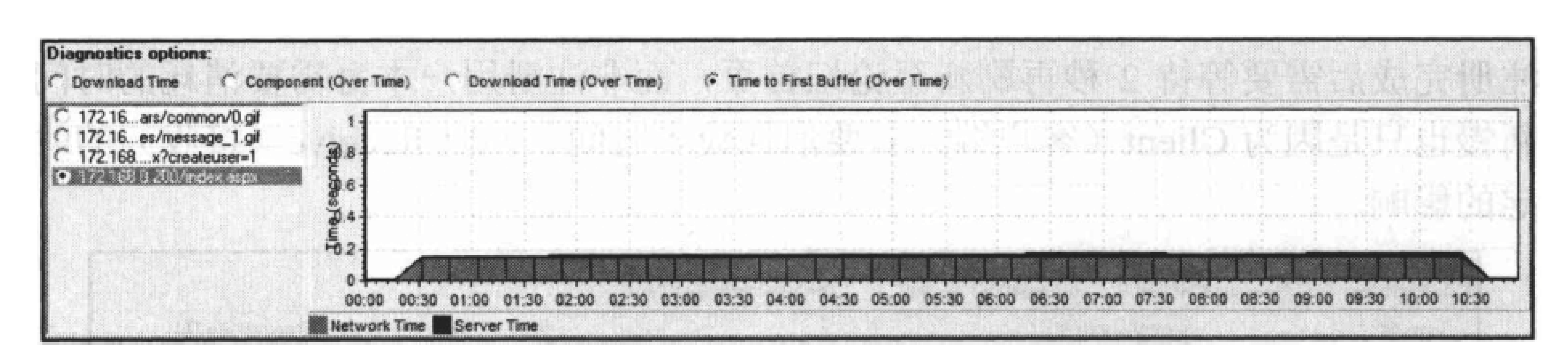


图 16-27　WebPage Diagnostics Time to First Buffer(Over time)

通过这 4 张分析图，可以了解到对于事务的响应时间来说，服务器的处理时间并不是组成响应时间的主要部分，而网络问题通常会占用超过 70% 的时间，通过 Web Page Diagnostics 可以准确分析响应时间，避免由于网络延迟或者带宽问题而影响对响应时间的分析和瓶颈定位。

② Page Download Time Breakdown(页面响应时间组成分析)。

这张图中显示了每个页面响应时间的组成分析，一个页面的响应时间一般由以下内容组成。

a. Client Time：客户端浏览器接收所需要使用的时间，可以不用考虑。

b. Connections Time：连接服务器所需要的时间，越小越好。

c. DNS Resolution Time：通过 DNS 服务器解析域名所需要的时间，解析受到 DNS 服务器的影响，越小越好。

d. Error Time：服务器返回错误响应时间，这个时间反映了服务器处理错误的速度，一般是 Web 服务器直接返回的，包含了网络时间和 Web 服务器返回错误的时间，该时间越小越好。

e. First Buffer Time：连接到服务器，服务器返回第一个字节所需要的时间，反映了系统对于正常请求的处理时间开销，包含网络时间和服务器正常处理的时间，该时间越小越好。

f. FTP Authentication Time：FTP 认证时间，这是进行 FTP 登录等操作所需要消耗的认证时间，越短越好。

g. Receive Time：接收数据的时间，这个时间反映了带宽的大小，带宽越大，下载时间越短。

h. SSL Handshaking Time：SSL 加密握手的时间。而 Analysis 在这里会分析得到页面请求的组成比例图，便于分析页面时间浪费在哪些过程中。在图 16-28 中可以看到各个页面请求的响应时间组成情况，相对于 172.168.0.200 的首页请求，时间主要浪费在 Client 上。

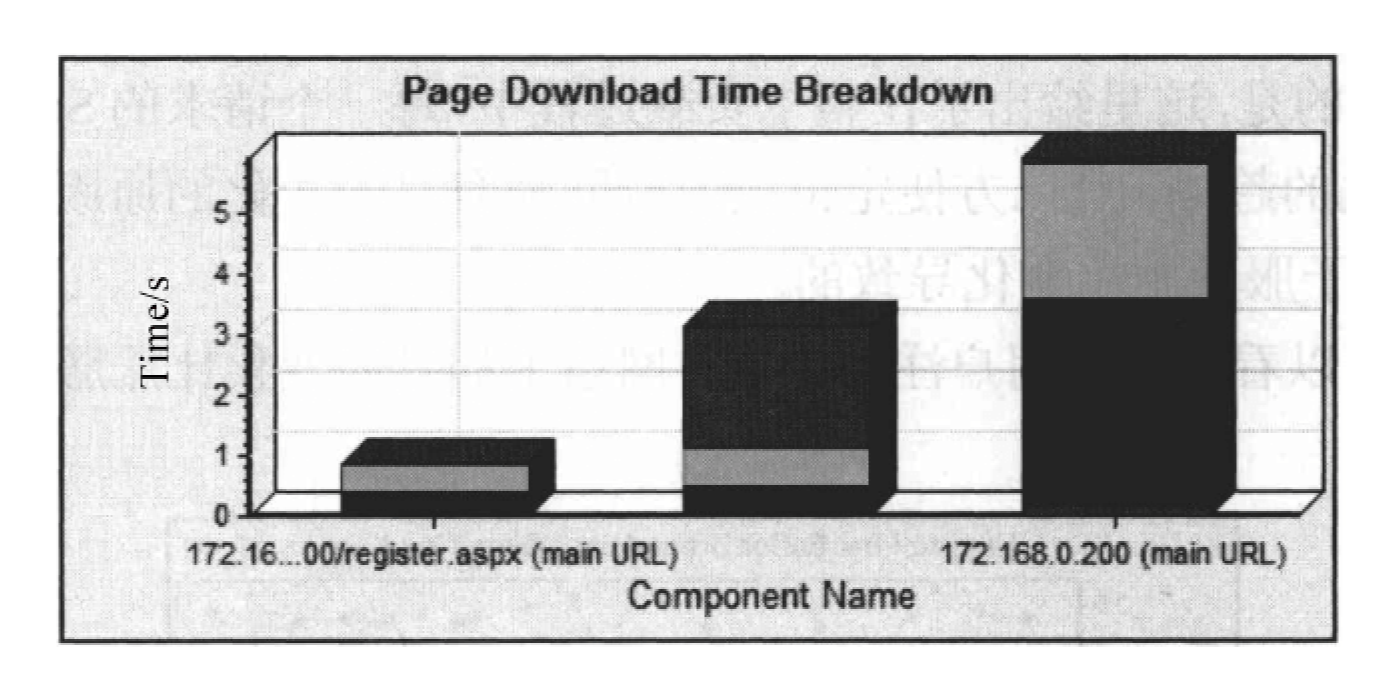


图 16-28　Page Download Time Breakdown

③ Page Download Time Breakdown(Over Time)(页面组成部分时间)。

这里提供了随着时间的变化所有请求的响应时间变化过程。这里会将整个负载过程中每个页面的每个时间组成部分都做成单独的时间线，以便分析在不同的时间点上组成该页面的各个请求时间是如何变化的。在图 16-29 中可以看到大多数页面的响应时间是比较稳定的，其中首页刷新变动较大。

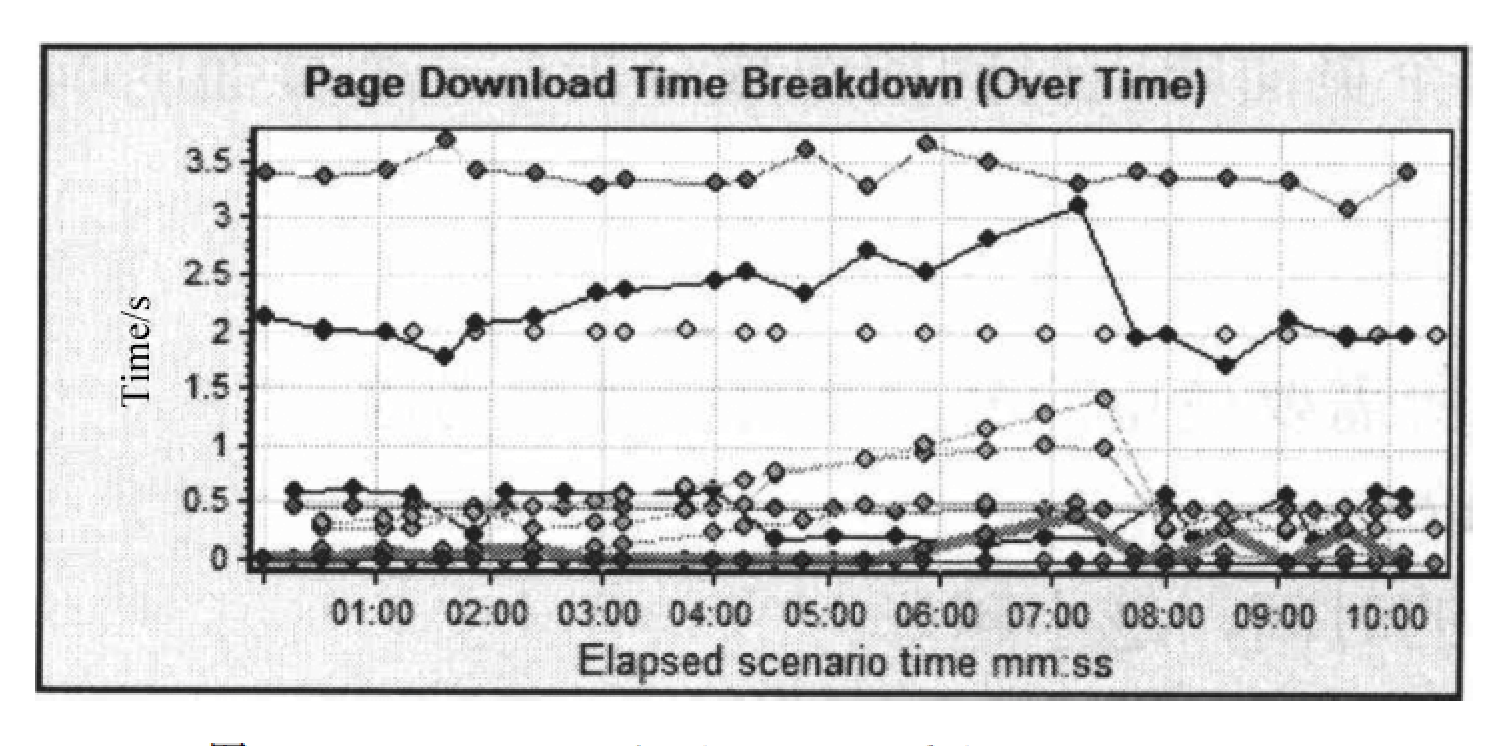


图 16-29　Page Download Time Breakdown(Over Time)

首先找到变化最明显或者响应时间最高的页面，随后再针对这个页面进行进一步的分析了解时间偏长或者变化较快的原因。

④ Time to First Buffer Breakdown（页面请求组成时间）。

这里提供了组成页面时间请求的比例说明（客户端时间/服务器时间），通过这张图，可以直观地了解到整个页面的处理是在服务器端消耗的时间长，还是在客户端消耗的时间长，从而分析得到系统的性能问题是在前端还是在后端。

在图16-30中可以看出对于整个负载来说，网络或客户端的时间开销占了绝大多数。

⑤ Time to First Buffer Breakdown（Over Time）（基于时间的页面请求组成分析）。

和图16-30不同的是，这里给出了在整个负载过程中，每一个请求的Server Time和Client Time随着时间变化的趋势，可以方便定位响应时间随着时间变化的原因到底是由于客户端变化导致的还是由于服务器端变化导致的。

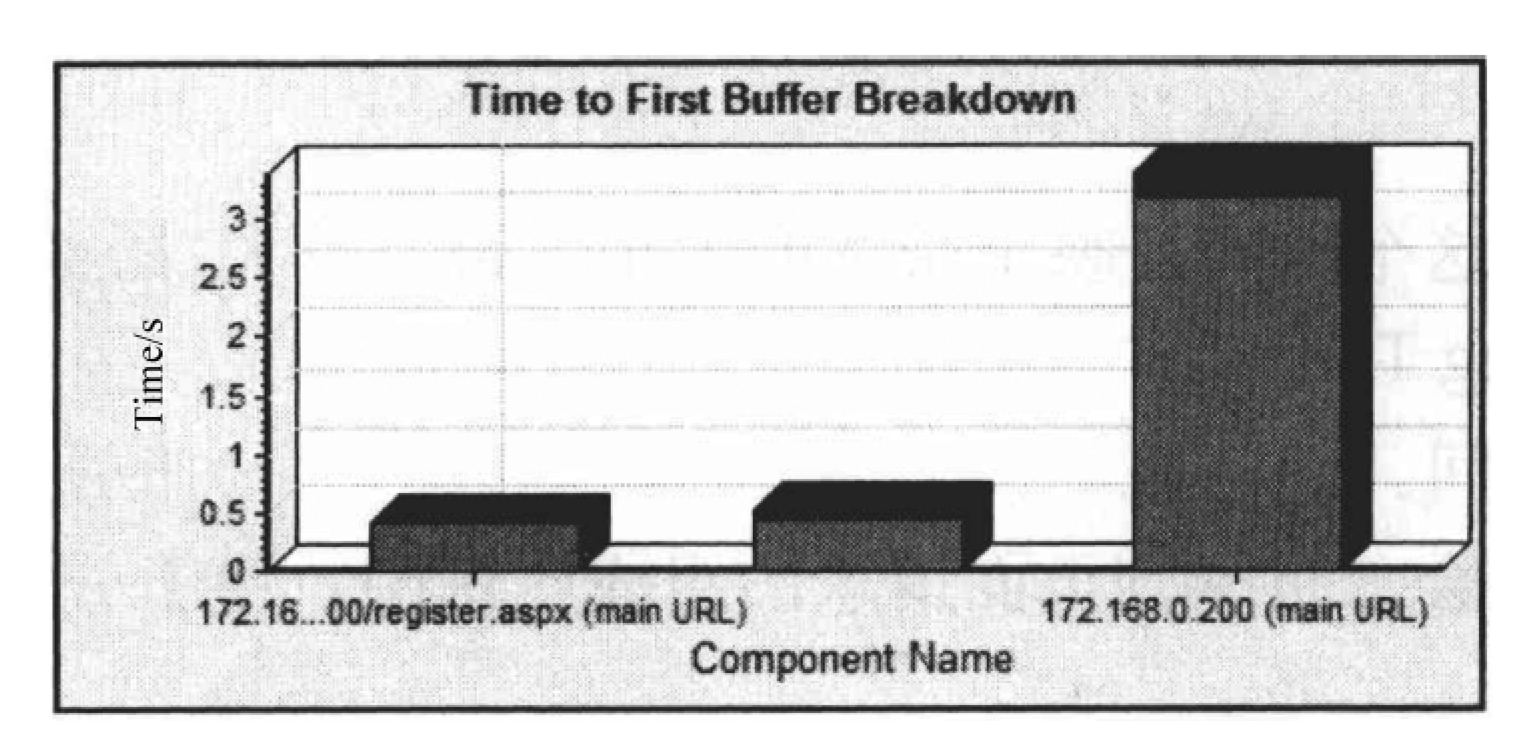


图16-30 Time to First Buffer Breakdown

在图16-31中可以看到对于用户注册操作，网络上的时间变化比服务器上的时间变化要剧烈。

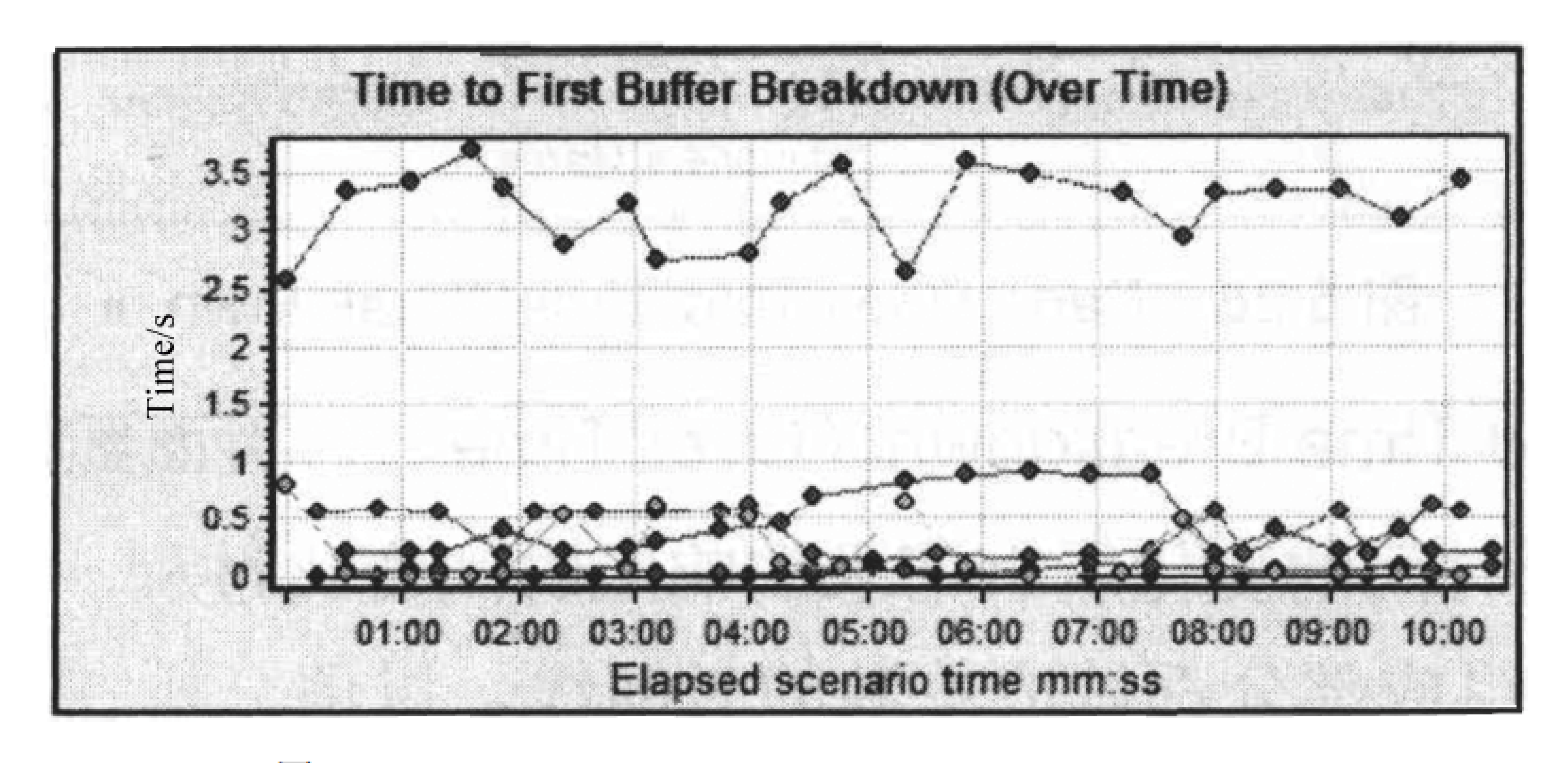


图16-31 Time to First Buffer Breakdown（Over Time）

（6）网络监控（Network Monitor）图：如果在Controller中添加了Network Delay Time监控后会出现该数据图。这个功能很好但并不是非常直观和方便，建议使用第三方专门的路由分析工具进行网络延迟和路径分析。

① Network Delay Time。

这里会给出从监控机至目标主机的平均网络延迟变化情况。在图 16-32 中可以看到网络延迟从 240ms 逐渐减少到 26ms，最后上升到 340ms。

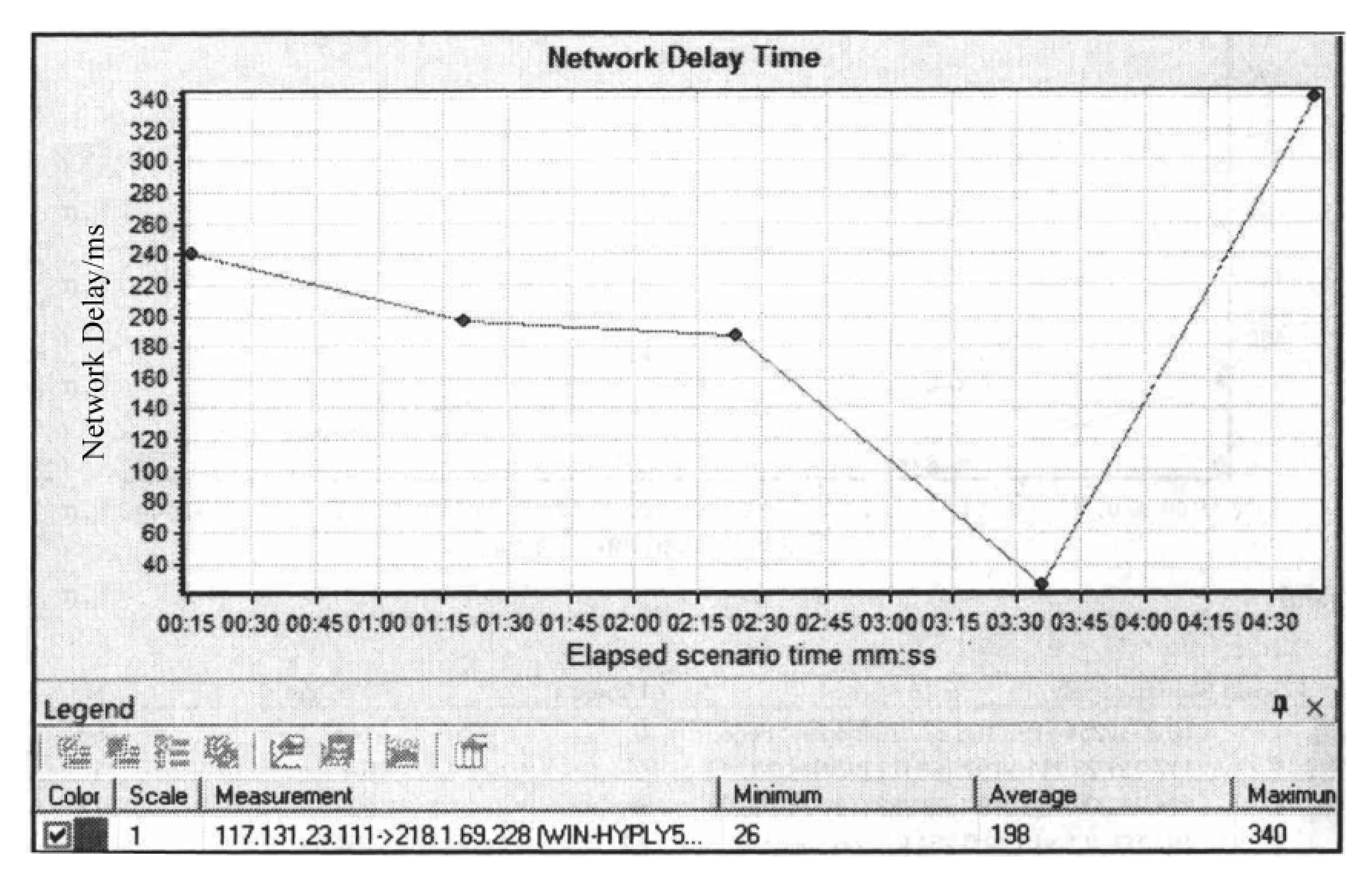


图 16-32 Network Delay Time

② Network Sub-Path Time。

这里给出从监控机至目标机各个网络路径的平均时间。当客户端在连接一个远程服务器时，路径并不是唯一的，受到路由器的路由选择，可能会选择不同的路径最终访问到服务器。在图 16-33 中列出了从监控服务器至目标服务器所经历的路径，以及每个路径上的网络延迟。

③ Network Segment Delay Time。

这里给出各个路径上各个节点的网络延迟情况。和图 16-33 不同的地方在于，这里给出的是路由器和路由器之间的网络延迟情况，针对连接而不是路径。

图 16-34 中给出了路由器和路由器之间的网络延迟变化情况，以便于分析影响整个网络时间的原因及节点。

(7) 资源监控(System Resources)图：资源包括很多种，在 Analysis 中监控的都是各种系统的计数器，这些计数器反映了系统中硬件或者软件的运行情况，通过它可以发现系统瓶颈。要想获得系统资源图，必须预先指定相关的计数器。

① System Resources(系统资源)。

这里给出了对操作系统计数器的监控，列出了在负载过程中系统的各种资源数据是如何变化的，该图需要在场景中设置了对应系统的监控后才出现。

② Database Server Resources(数据库资源)。

这里给出了数据库的相关资源在负载过程中的变化情况。

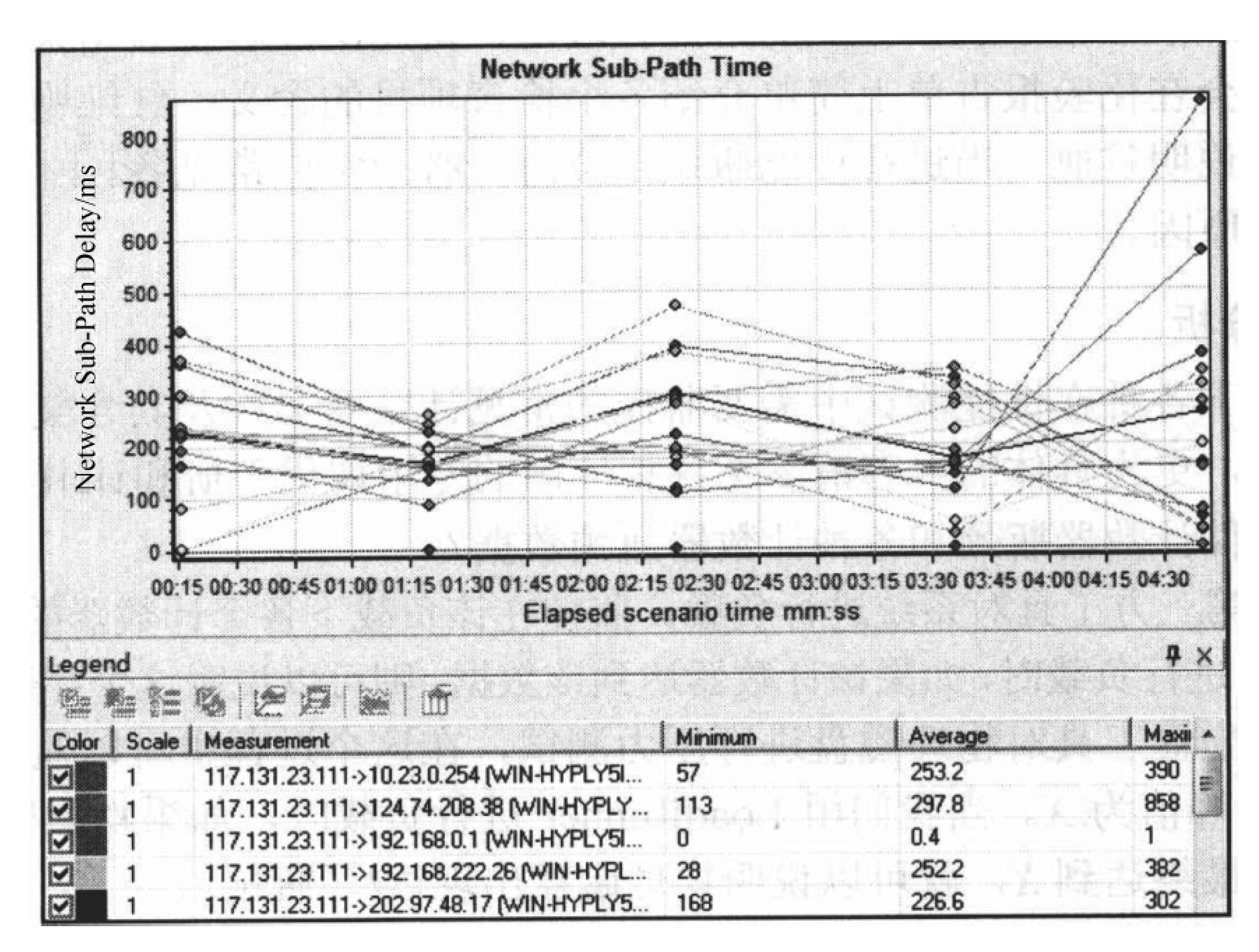


图 16-33 Network Sub-Path time

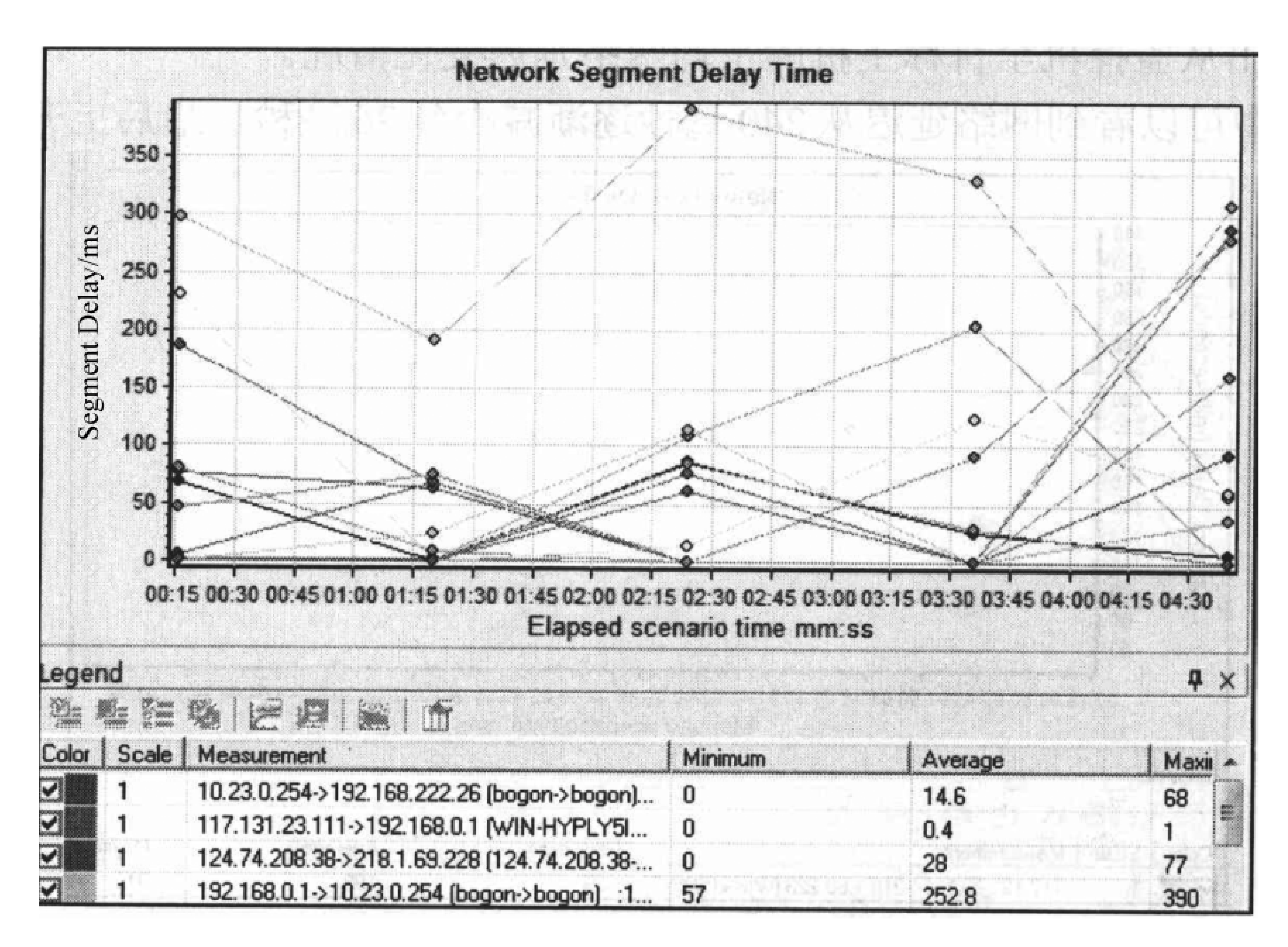


图 16-34 Network Segment Delay Time

③ Web Server Resources(Web 服务器资源)。

这里给出了 Web 服务器资源在负载过程中的变化情况。

面对 Analysis 提供的几十个测试结果分析图，很多人会感到无所适从，不知如何入手。实际上，性能测试分析要求执行人员更加谨慎和细心，不能放过任何一个缺陷，尤其要深入系统内部来进行分析。同时，当分析结果时还应该借助 Analysis 以外的各种分析工具。例如，可以借助 Oracle 提供的监控与分析工具，也可以借助 WebLogic 提供的监控与分析工具，要想尽一切办法来发现系统瓶颈。

下面介绍一些通用的性能测试分析流程。

第一步：从分析 Summary 的事务执行情况入手。

Summary 主要是判定事务的响应时间与执行情况是否合理。如果发现问题，则需要做进一步分析。通常情况下，如果事务执行情况失败或响应时间过长等，都需要做深入分析。

下面是查看分析概要时的一些原则：

用户是否全部运行，最大运行并发用户数（Maximum Running Vusers）是否与场景设计的最大运行并发用户数一致。如果没有，则需要打开与虚拟用户相关的分析图，进一步分析虚拟用户不能正常运行的详细原因。

事务的平均响应时间、90%事务最大响应时间用户是否可以接受。如果事务响应时间过长，则要打开与事务相关的各类分析图，深入地分析事务的执行情况。

查看事务是否全部通过。如果有事务失败，则需要深入分析原因。很多时候，事务不能正常执行意味着系统出现了瓶颈。

如果一切正常，则本次测试没有必要进行深入分析，可以进行加大压力测试。

如果事务失败过多，则应该降低压力继续进行测试，使结果分析更容易进行。

上面这些原则都是分析 Summary 的一些常见方法，读者应该灵活使用并不断地进行总结与完善，尤其要注意结合实际情况，不能墨守成规。

第二步：查看负载发生器和服务器的系统资源情况。

查看分析概要后，接下来要查看负载发生器和待测服务器的系统资源使用情况：查看 CPU 的利用率和内存使用情况，尤其要注意查看是否存在内存泄漏问题。这样做是由于很多时候系统出现瓶颈的直接表现是 CPU 利用率过高或内存不足。

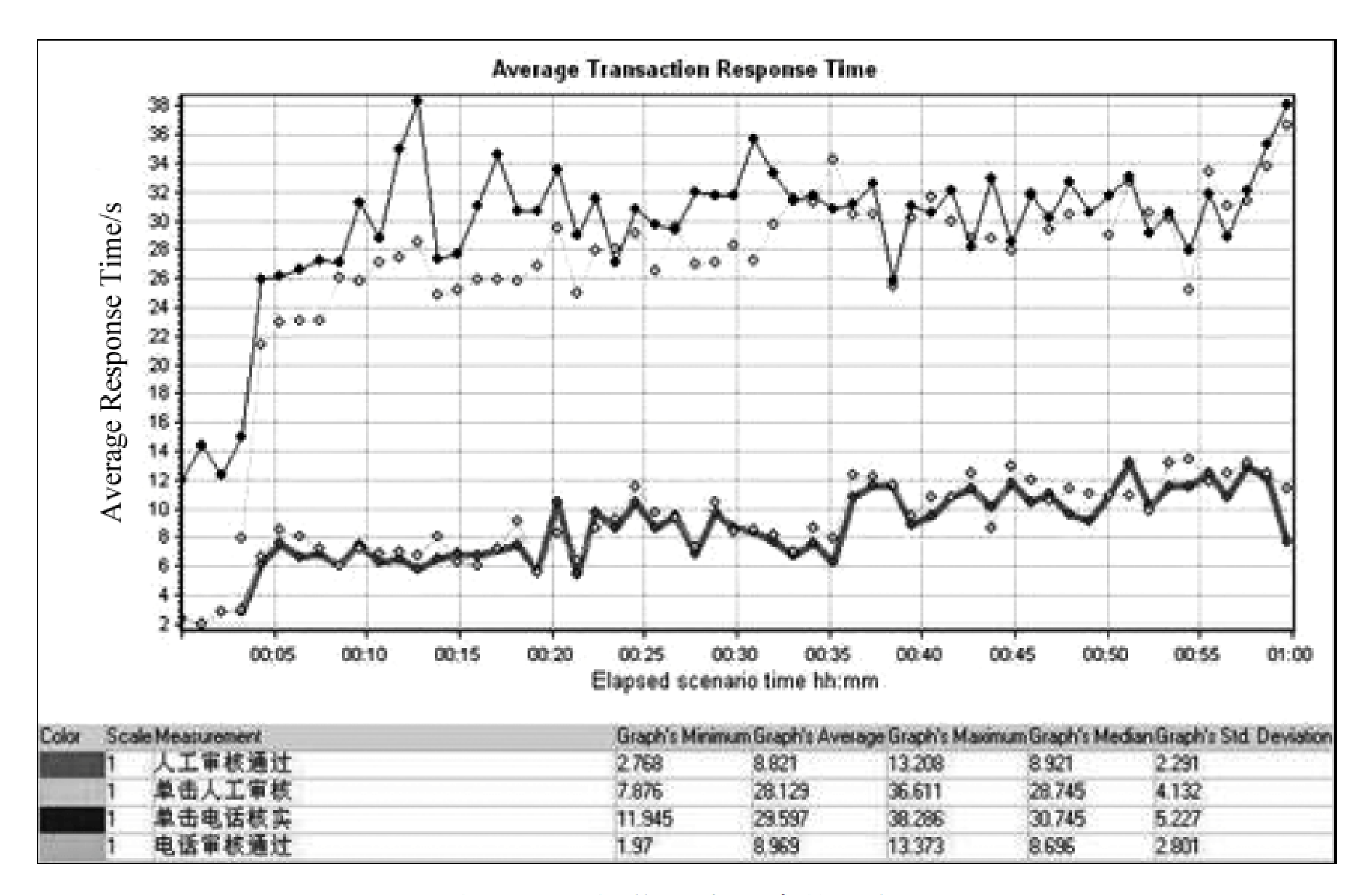
应该保证负载发生器在整个测试过程中其 CPU、内存、带宽没有出现瓶颈，否则测试结果无效。而待测试服务器，则重点分析测试过程中 CPU 和内存是否出现了瓶颈。CPU 需要查看其利用率是否经常达到 100%或平均利用率一直高居 95%以上；内存需要查看是否够用以及测试过程是否存在溢出现象（对于一些中间件服务器要查看其分配的内存是否够用）。

第三步：查看虚拟用户与事务的详细执行情况。

在前两步确定了测试场景的执行情况基本正常后，接下来就要查看虚拟用户与事务的执行情况。对于虚拟用户，主要查看在整个测试过程中是否运行正常，如果有较多用户不能正常运行，则需要重新设计场景或调整用户加载与退出方式再次进行测试。对于事务，重点关注整个过程的事务响应时间是否逐渐变长以及是否存在不能正常执行的事务。

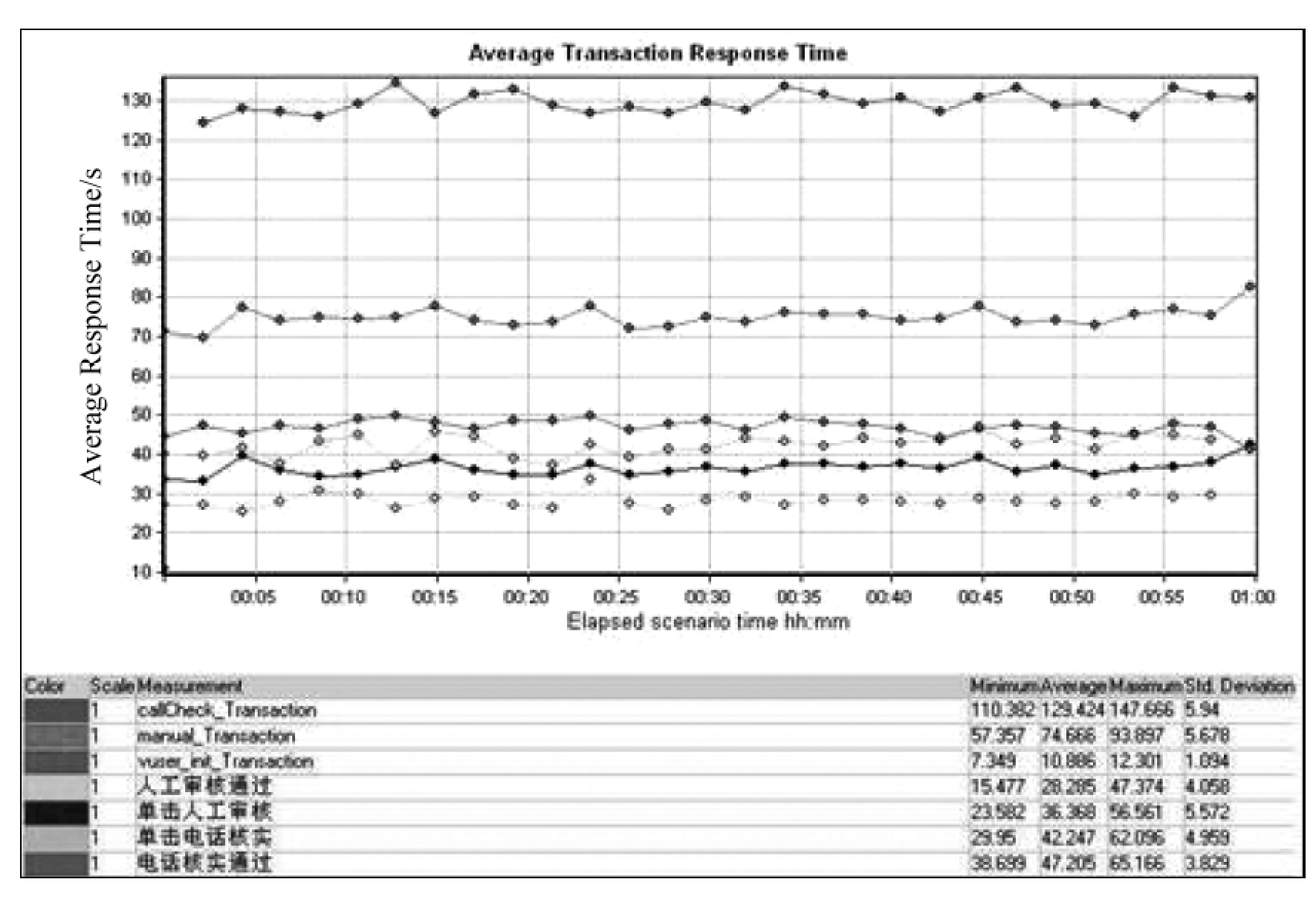
总之，任何用户或事务的执行细节都应该认真分析，不可以轻易忽略。如图 16-35 所示的就是一个性能逐步下降的服务器，需要进一步分析其性能下降的原因，例如查找是否

存在内存泄漏问题;图 16-36 则是一个性能相对稳定的服务器,但是响应时间偏大,这时需要分析程序算法是否存在缺陷或服务器参数的配置是否合理。



Color	Scale	Measurement	Graph's Minimum	Graph's Average	Graph's Maximum	Graph's Median	Graph's Std. Deviation
	1	人工审核通过	2.768	8.821	13.208	8.921	2.291
	1	单击人工审核	7.876	28.129	36.611	28.745	4.132
	1	单击电话核实	11.945	29.597	38.286	30.745	5.227
	1	电话审核通过	1.97	8.969	13.373	8.696	2.801

图 16-35　性能逐步下降的服务器



Color	Scale	Measurement	Minimum	Average	Maximum	Std. Deviation
	1	callCheck_Transaction	110.382	129.424	147.666	5.94
	1	manual_Transaction	57.357	74.666	93.897	5.678
	1	vuser_init_Transaction	7.349	10.886	12.301	1.094
	1	人工审核通过	15.477	28.285	47.374	4.058
	1	单击人工审核	23.582	36.368	56.561	5.572
	1	单击电话核实	29.95	42.247	62.096	4.959
	1	电话核实通过	38.699	47.205	65.166	3.829

图 16-36　性能稳定的服务器

下面是虚拟用户与事务分析的常用准则:

虚拟用户如有失败,则要查明原因。

在整个测试过程中,所有的虚拟用户是否一直稳定运行并成功执行全部事务。如果

仅有一个用户或部分用户能够正常运行，则说明测试脚本可能存在问题。

对于失败的事务首先要分析其失败原因，接着要查看事务的失败是否导致了用户失败。

判断用户是否可以接受事务平均响应时间值以及 90%用户的最大响应时间值。

查看整个测试过程的事务平均响应时间是否逐步变大，正常情况下，事务平均响应时间的变化应该是接近于平行 X 轴的一条直线。

事务响应时间是否在整个测试过程中随着用户的增加而线性变短。正常情况应该是，当一定范围内的用户并发时，事务响应时间应不会有太大的变化。

服务器每秒通过的事务总数、某一事务每秒通过数是否稳定，如果整个测试过程基本不变，则要分析是服务器达到了处理上限，还是 Generator 产生的压力达到了上限。

按照迭代次数来运行的场景，要分析通过的事务总数是否与设定的一致。如果不一致，则可能是测试脚本存在错误，也可能是待测试程序存在功能错误，应该在调整后再次进行测试。

Analysis 对虚拟用户和事务提供了非常强大的跟踪功能，可以跟踪每一个用户及其相关事务的执行情况。

第四步：查看错误发生情况。

整个测试过程的错误发生情况是分析的重点。下面查看错误发生情况的常用准则：

查看错误发生曲线在整个测试过程中是否有规律变化，如果是，则意味着程序在并发处理方面存在一定的缺陷。如图 16-37 所示的每秒缺陷数量曲线很有规律，这是因为服务器定期生成缓存文件导致用户不能正常访问而产生的错误；查看错误分类统计，作为优化系统的参考。例如 Web 性能测试，当出现瓶颈时往往需要查看服务器的错误统计信息结果：如果“超时错误”达到 90%以上，可能需要提高硬件配置；如果有较多的“内部服务器错误”，则可能是程序方面存在问题。

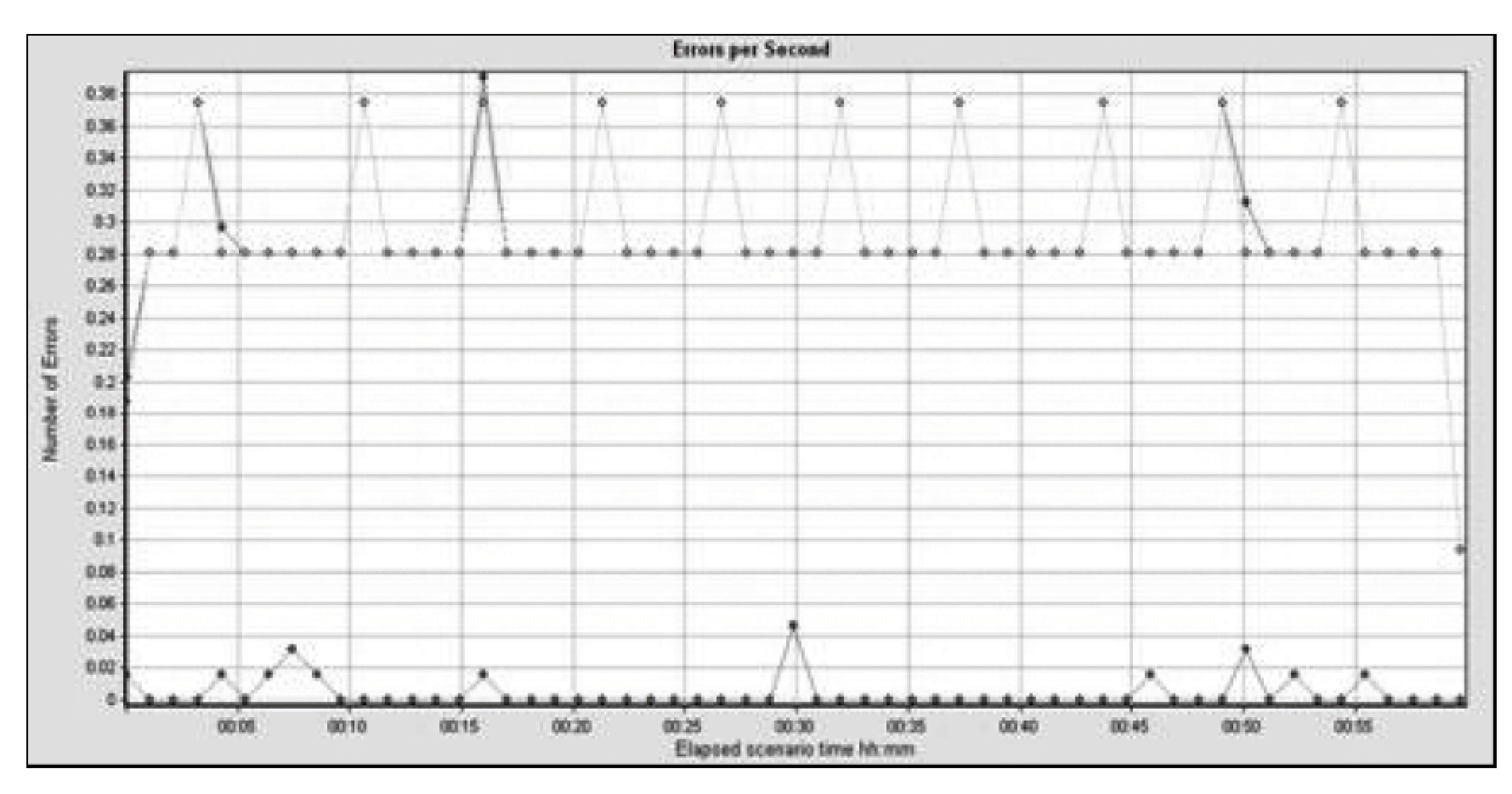


图 16-37 规律变化的每秒错误数曲线图

第五步：查看 Web 资源与细分网页。

本步骤仅适用于 Web 性能测试。查看 Web 资源图时，往往需要结合前面对虚拟用

户以及事务响应时间的分析结果,重点分析服务器的稳定性。对于网页细分功能则应遵循以下原则：首先分析从用户发出请求到收到第一个缓冲为止,哪些环节比较耗时;其次找出页面中哪些组成部分对用户响应时间影响较大;在页面的性能问题定位后,就可以采取相关的解决方案。

16.3 通过 LR 图表组合挖掘系统缺陷根源

(1) Merge Graphs(合并图)。

当得到了相关图形,如何分析图和图之间的关系呢？Merge Graphs 提供了一种将图和图合并的功能,通过这个功能可以直观地获得一个数据和另外一个数据的相互影响关系。

在 Running Vusers 图中,单击鼠标右键,在菜单中选择 Merge Graphs,如图 16-38 所示。

Merge 的方式有三种：Overlay、Tile、Correlate,如图 16-39 所示。

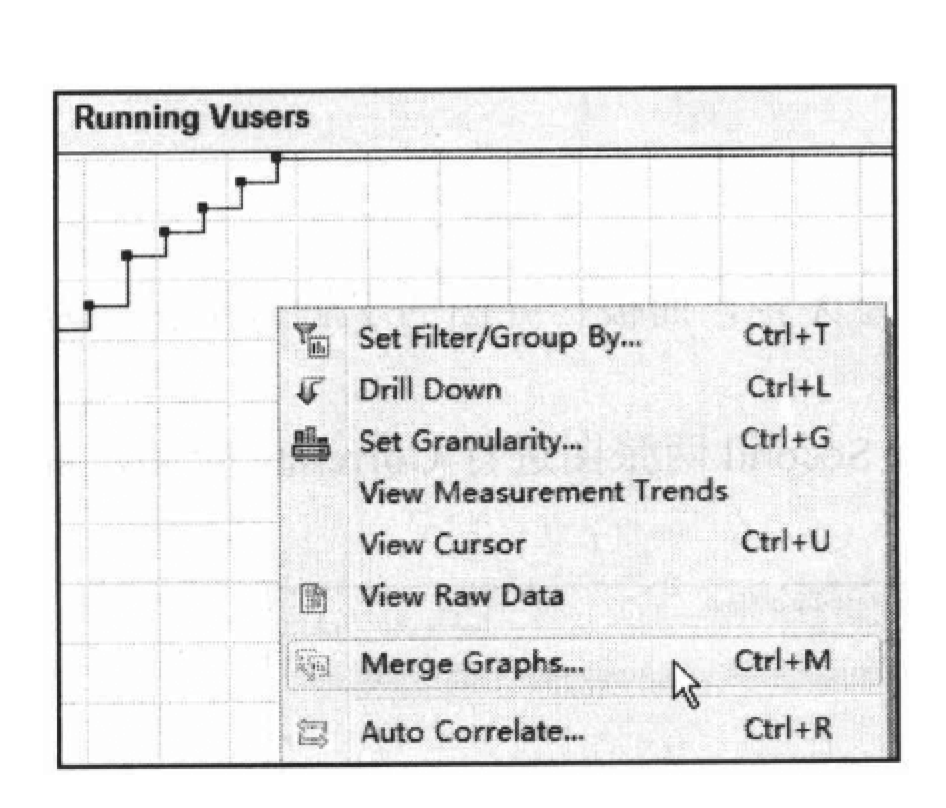


图 16-38 Merge Graphs

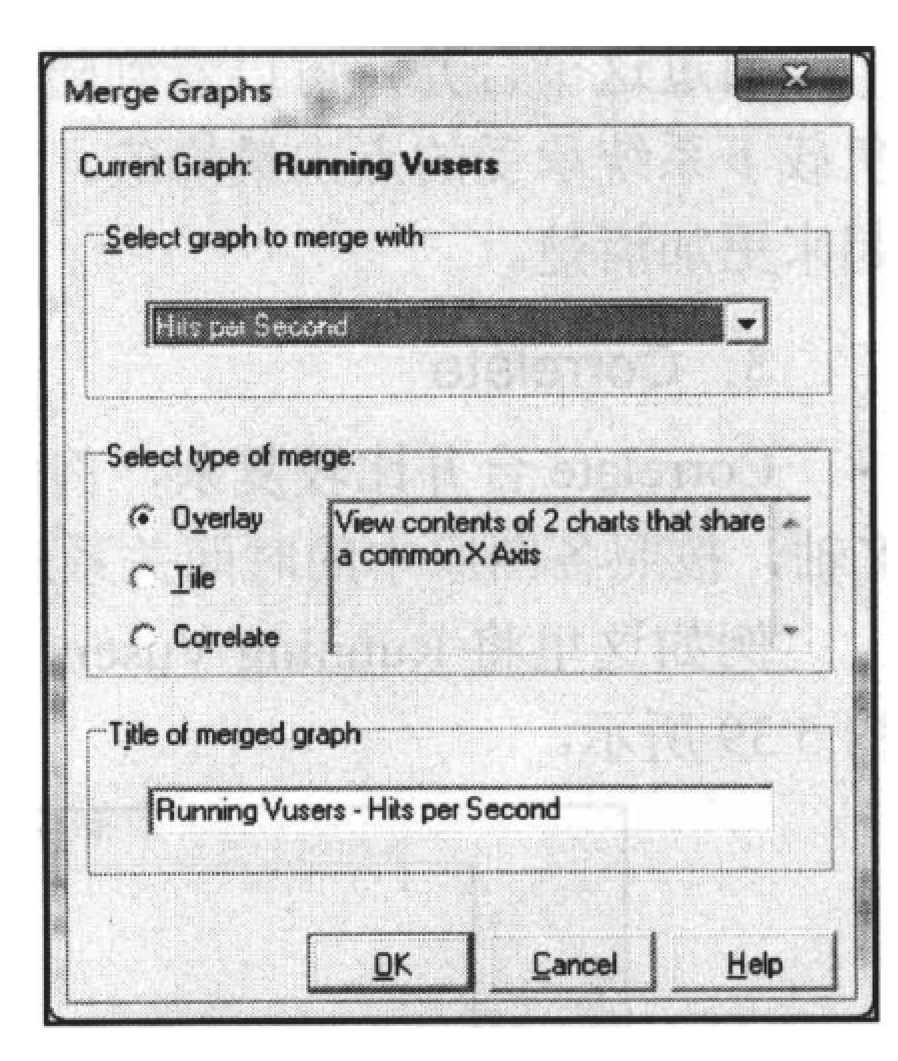


图 16-39 Merge 方式设置

这三种合并方法可以帮助人们对测试结果进行分析,了解图和图之间的影响关系。

① Overlay。

基于 Overlay 合并方式,就是将两张图通过 X 轴进行覆盖合并。例如将 Running Vusers 和 Average Transaction Response Time 进行 Overlay 合并,如图 16-40 所示。

通过这张图可以得到用户增长的过程是如何影响事务平均时间的,从而发现瓶颈出现的时间。

② Tile。

Tile 模式和 Overlay 方式非常接近,只是将两张图的 Y 轴分了上下两部分,不做重叠。

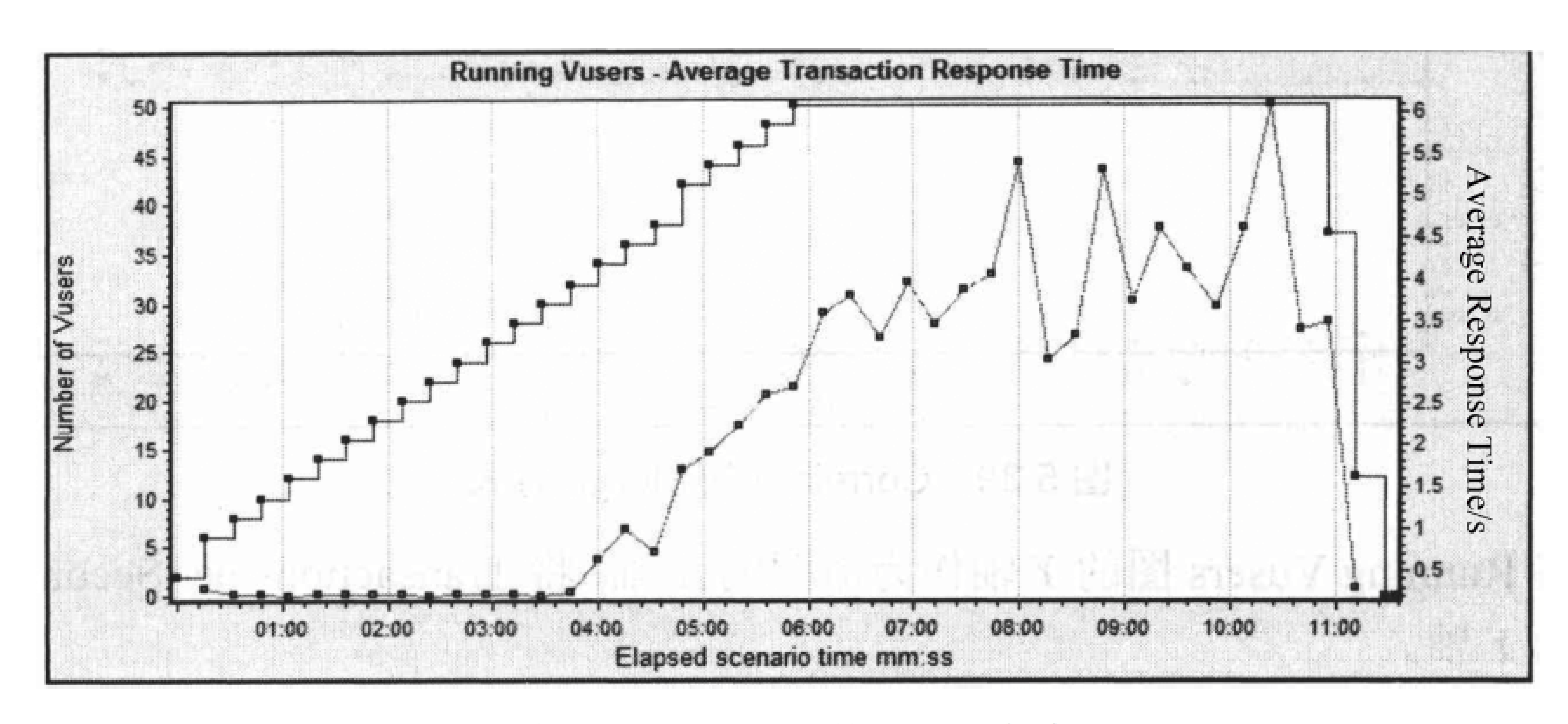


图 16-40　Overlay 的 Merge 方式

将 Running Vusers 和 Hits per Second 进行 Tile 合并，如图 16-41 所示。

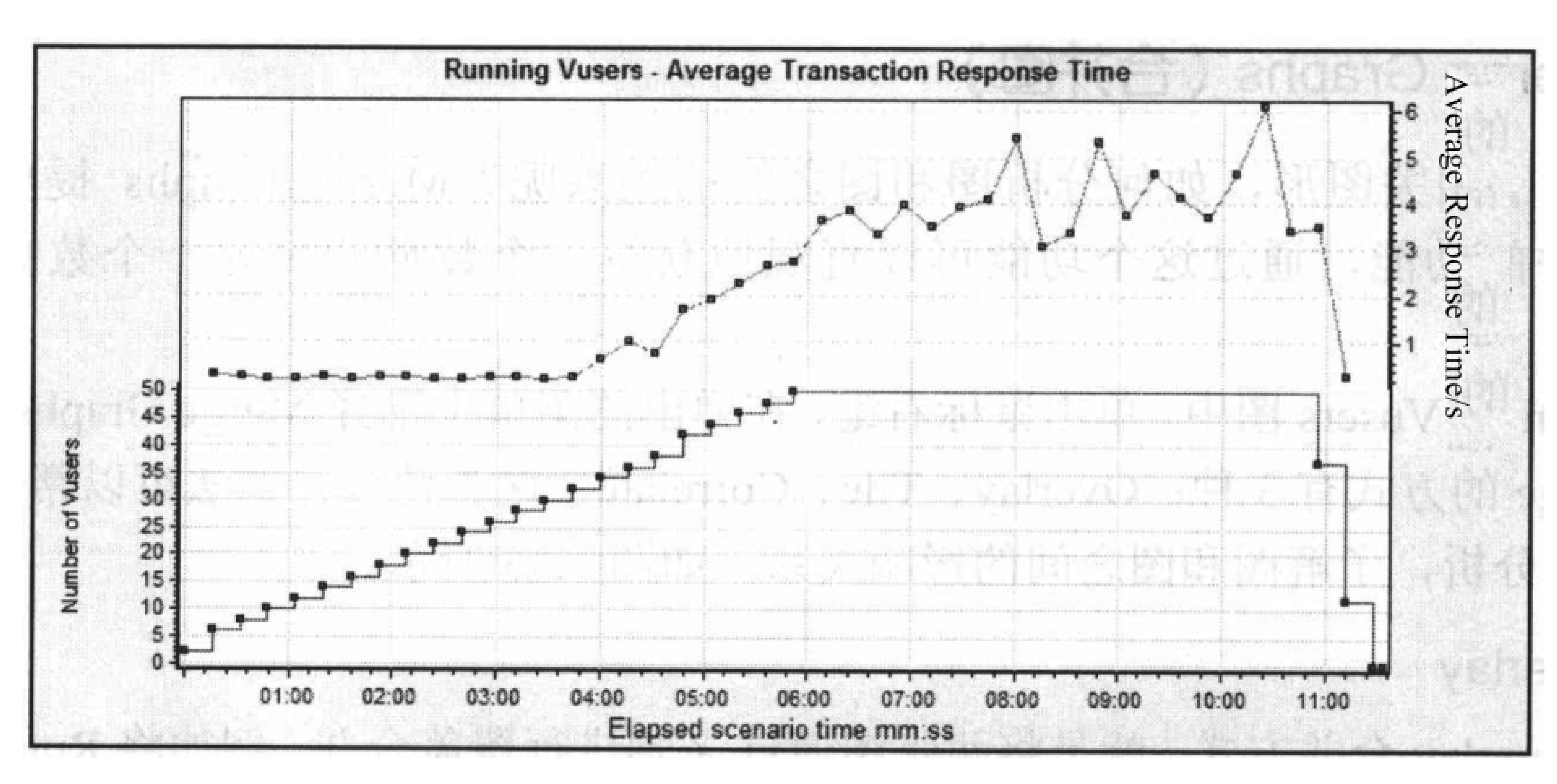


图 16-41　Tile 的 Merge 方式

通过这个合并，可以看到随着用户数量增加每秒单击量的变化过程，从而了解在当前负载下系统承受的单击量峰值。相对于 Overlay 方式，两张图的线条不会重叠在一起，看起来更加清楚。

③ Correlate。

这个合并比较复杂，首先将主图的 Y 轴变为 X 轴，而被合并图的 Y 轴依然保存为 Y 轴，按照各图原本的时间关系进行合并形成新图。

例如将 Running Vusers 和 Transactions per Second 两张图进行 Correlate 合并，如图 16-42 所示。

系统将 Running Vusers 图的 Y 轴作为新图的 X 轴，将 Transactions per Second 图的 Y 轴作为新图的 Y 轴。在通过 Correlate 方式合并后的图中，可以更加清晰地看出用户变化导致处理能力的变化过程，斜率越大说明影响越大，方便快速定位在何种负载下系统响应时间能够稳定，而在何种负载下响应时间会大幅上升。

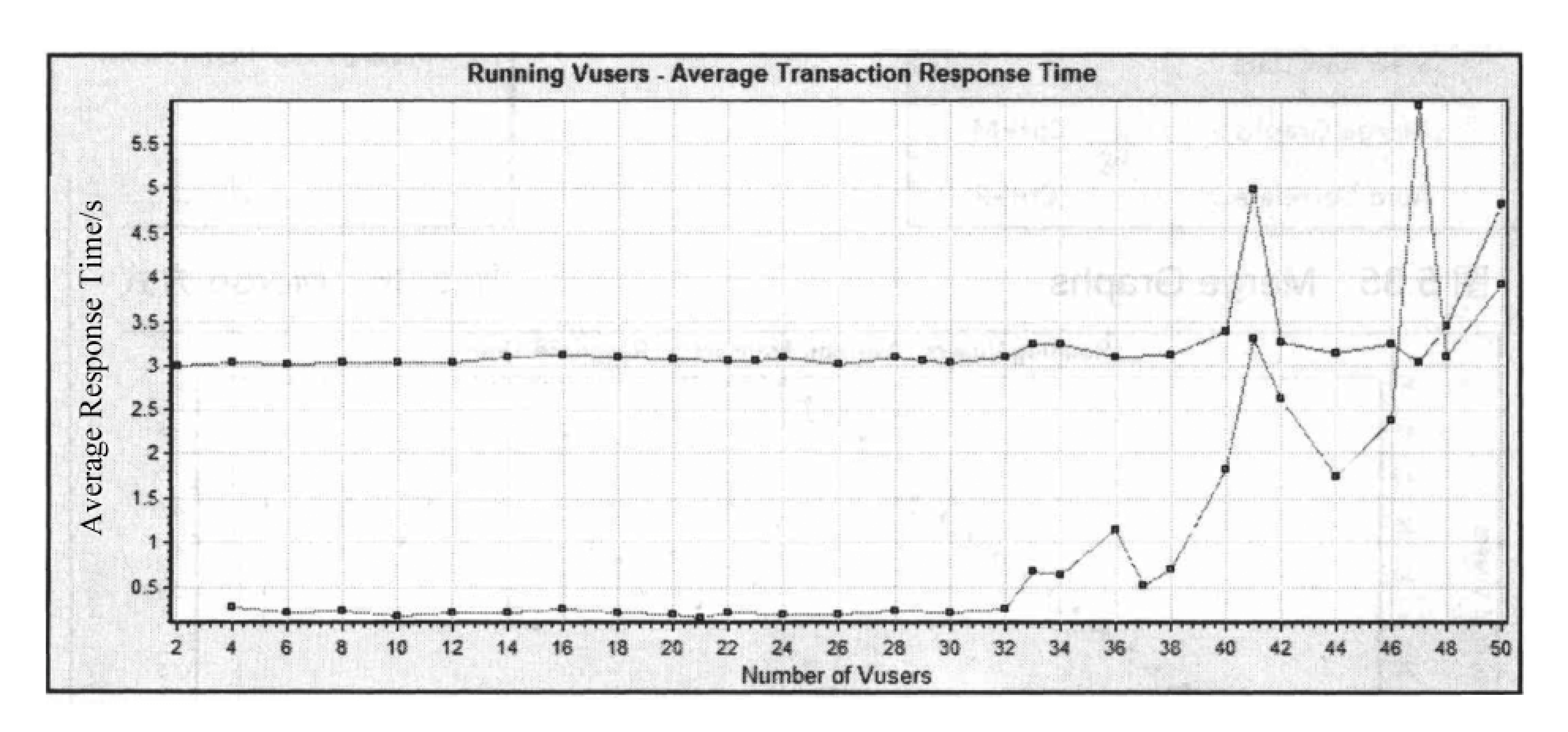


图 16-42 Correlate 的 Merge 方式

通过这几种常见的 Merge 方法,可以将相关计数器得到的图进行合并分析,找出系统性能的瓶颈。由于系统瓶颈的定位需要大量的相关知识,对于一个初级性能测试工程师来说,并不要求有性能结果分析和性能瓶颈定位的能力。初级性能测试工程师可以将数据整理后提交给项目经理、网络或数据库管理员,让他们帮助你分析数据,并确认及完成性能调优工作。

(2) Auto Correlate(自动定位瓶颈)。

Auto Correlate 提供了自动分析趋势影响的功能,通过它可以方便地找出哪些数据之间有明显的相互依赖性,通过图和图之间的关系确认系统资源和负载相互影响的关系。

首先找到需要自动关联的图,单击右键打开 Auto Correlate 菜单。例如选择从 Running Vusers 图上做自动关联,如图 16-43 所示。

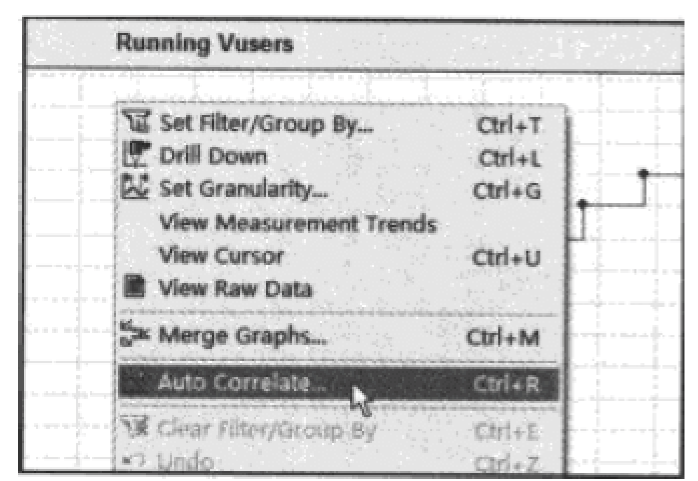


图 16-43 Auto Correlate

弹出自动关联窗口,如图 16-44 所示,需要进行以下设置。

① Time Range:设置关联的时间范围标签。

Suggest Time Range by 提供了自动关联的范围参考,可以选择 Trend 基于趋势或者选择 Feature 基于特征两种方式。这两种方式可以自动选择关联的范围,Trend 完整包含所有值得注意、分析的趋势,而 Feature 则是其间一个单独的片段。例如这里选择 Feature,如图 16-45 所示。

右侧的竖线就会换到 6min 的位置,因为从 0min 到 6min 是用户负载上升的阶段,这个阶段的特征就是增加。通过单击 Next 按钮可以切换到下一个特征,下一个特征是 6min40s 至 7min04s 的负载下降阶段,如图 16-46 所示。

可以通过拖动图中的两根竖线来确定关联的范围或直接修改 From 后的时间来调整关联的范围。这样就确认了希望关联的时间范围,接着需要设置被关联的对象。

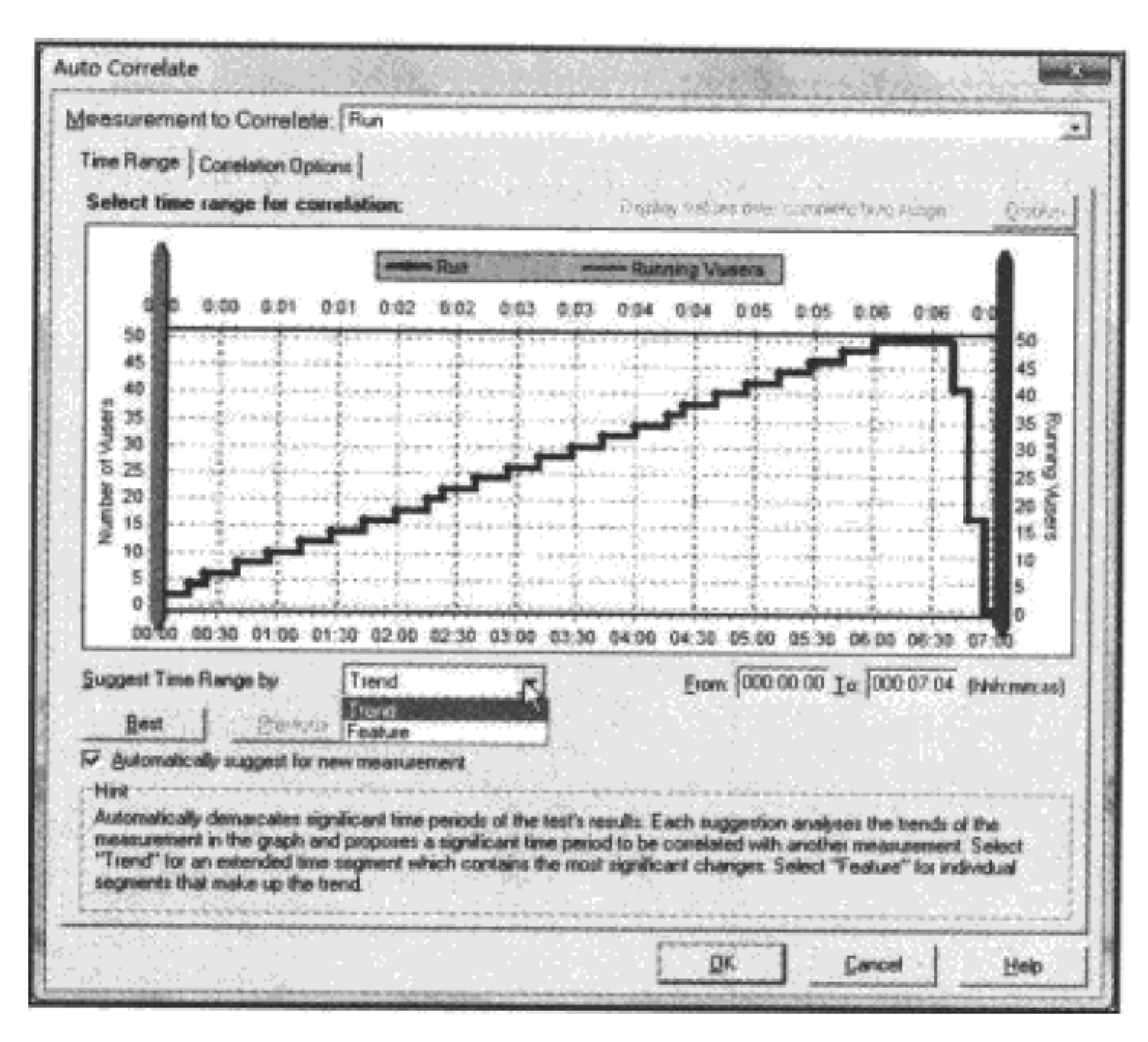


图 16-44　Auto Correlate 设置

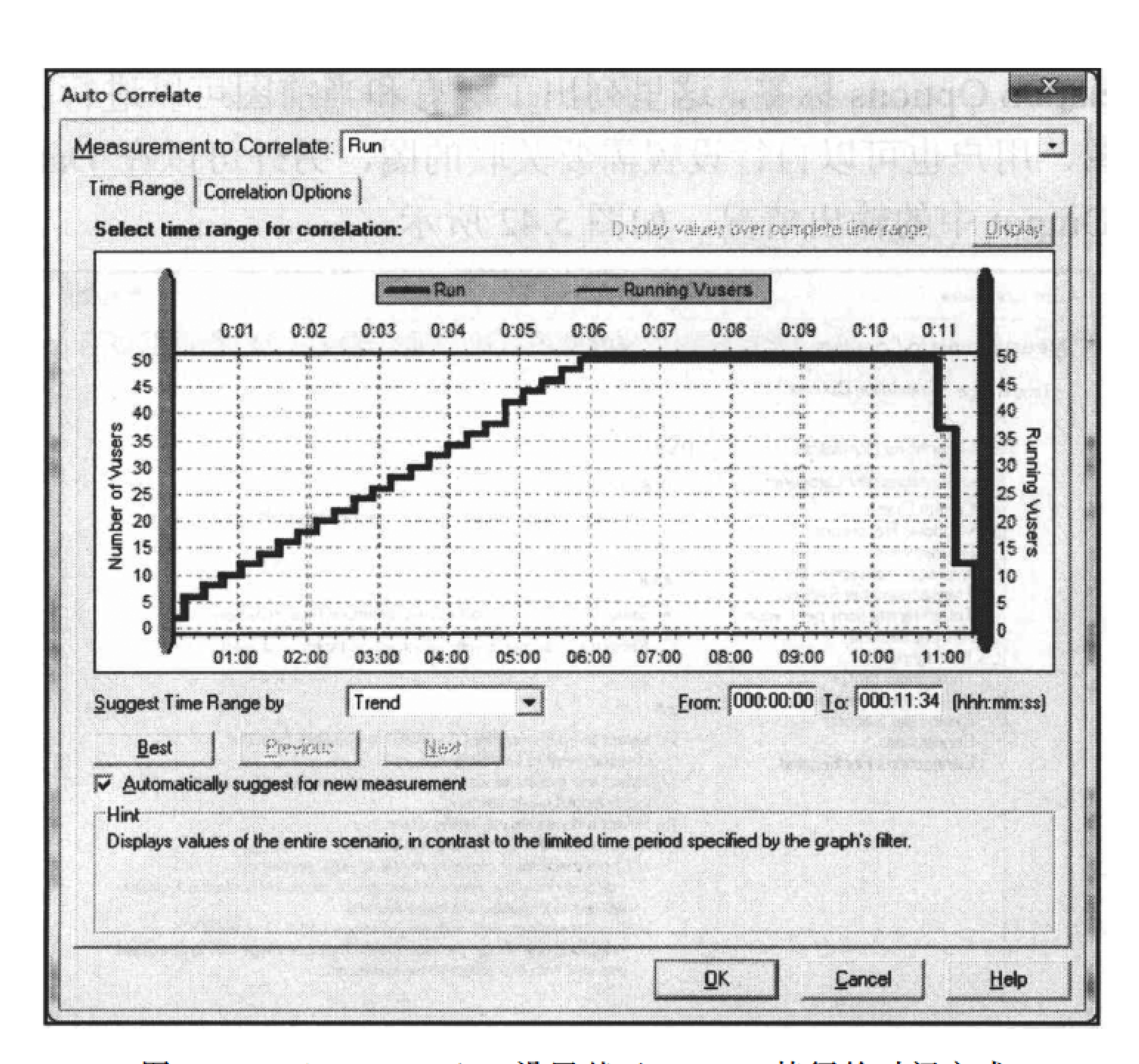


图 16-45　Auto Correlate 设置基于 Feature 特征的时间方式

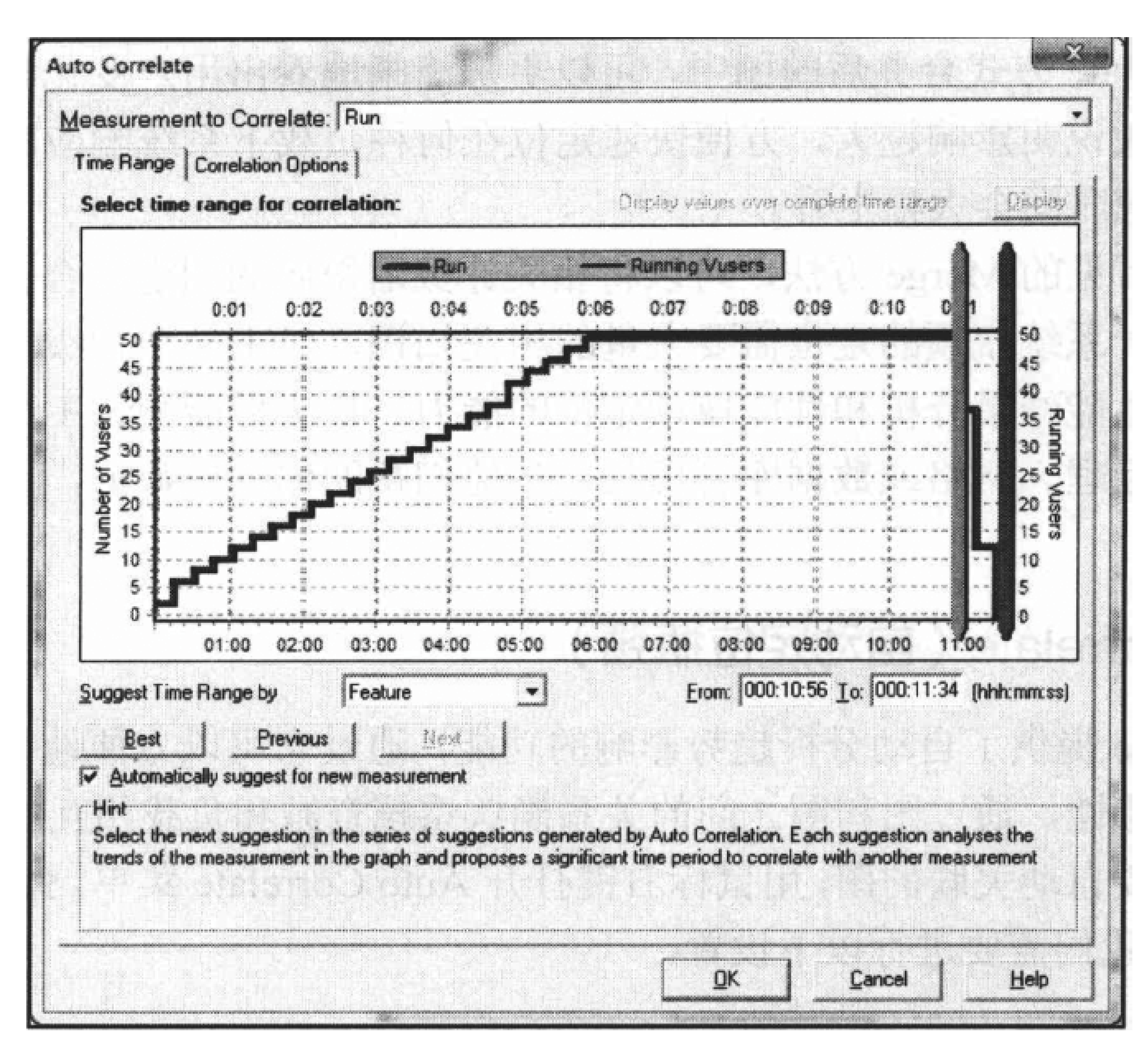


图 16-46　Auto Correlate 设置基于 Feature 特征的时间方式切换

② Correlation Options。

切换到 Correlation Options 标签，这里列出了所有和当前图可以进行关联的对象，默认选择了三个资源图，用户也可以自行设置需要关联的图，另外可以在 Data Interval 中设置数据点的间隔和 Output 中的输出情况，如图 16-47 所示。

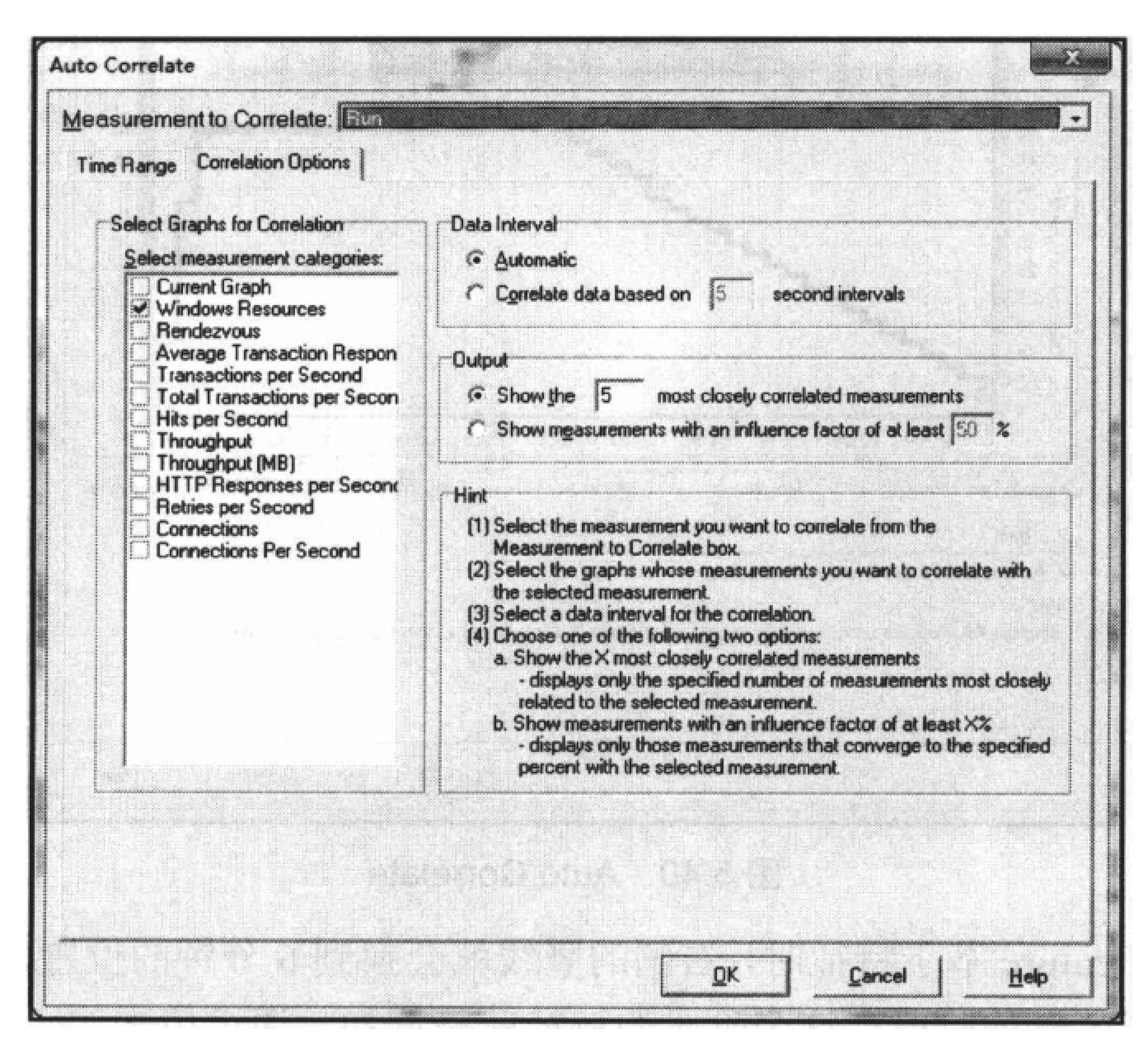


图 16-47　Auto Correlate 关联选项

Data Interval 是指自动关联的数据间隔，默认为 5min，也可以手工设定关联的数据间隔。间隔的时间设置越小得出的关联匹配度越精确。

Output 是对输出关联结果的设置，可以选择设置显示和被关联图匹配值最高的 5 个对象，也可以设置显示所有被关联图匹配值大于 50%的对象。

这里修改 Select measurement categories 选择对象为 Hits per Second，其他选项保持默认值，确定后得到最终的自动关联结果，如图 16-48 所示。

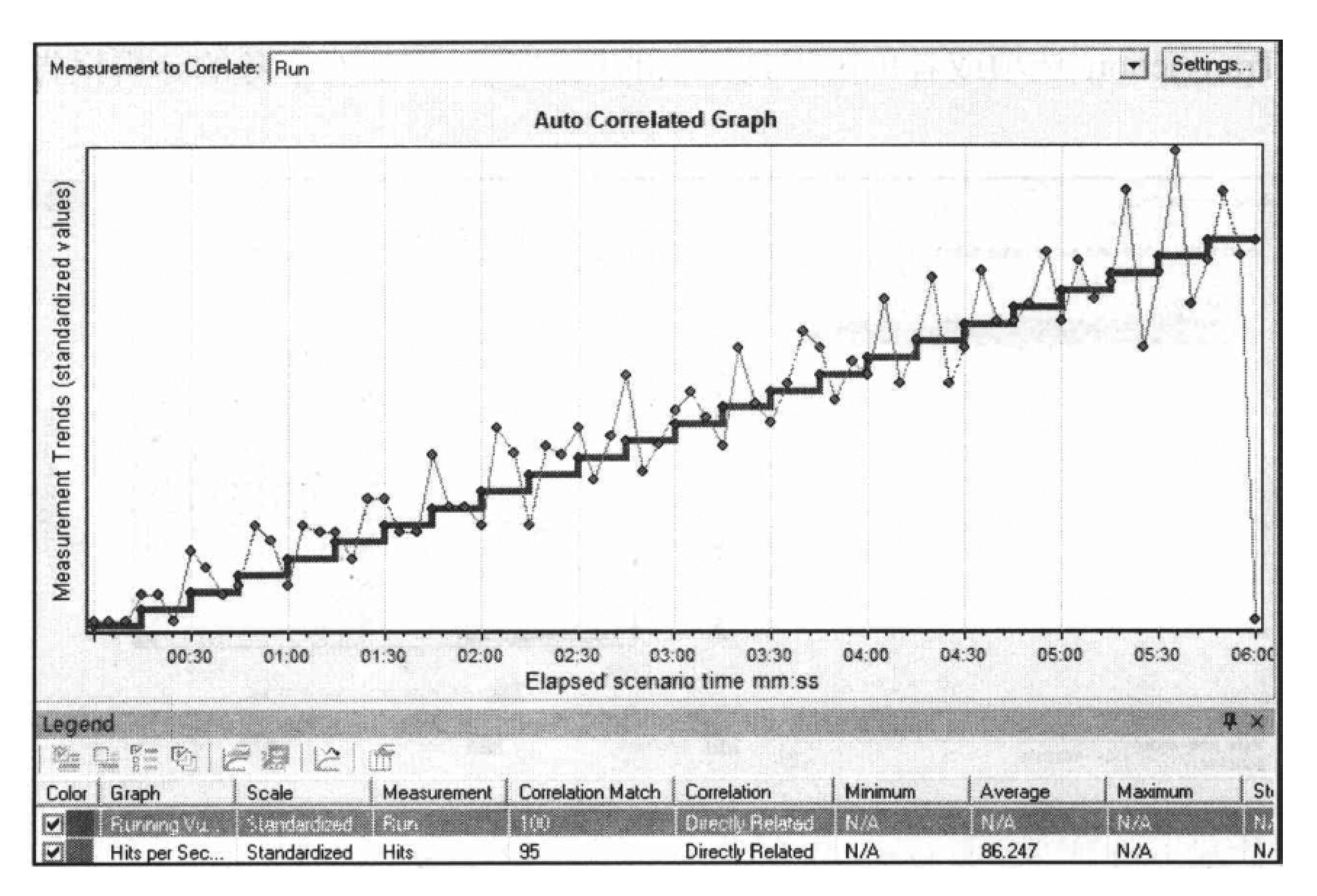


图 16-48 Auto Correlate 关联结果

在结果中可以看到用户数量增加和每秒单击量的关系，这和做一个 Run Vuser 和 Hits per Second 计数器的 overlay 型的 Merge 操作没什么区别。但在面的数据中可以看到 Collrelation Match 数据项，这个项目是自动关联独有的。Correlation Match 是指关联匹配度，该值反映了主数据和被关联数据的近似度。这里 Hits per Second 和被关联图 Running Vuser 的近似度为 95%，也就是说这两个数据变化的规律非常接近。

通过单击右上角的 Settings 按钮可以重新设置自动关联选项。通过设置 Correlation Options 下的 Data Interval 和 Output 来改变自动关联的结果，找到和被关联对象波动接近的对象，进一步分析各个数据间的影响关系，从而确认瓶颈产生的原因。

Auto Correlation 与 Merge 的区别在于：Merge 不能“选定特定的时间进行切片”，可先用 Merge 看整体趋势、分析全局。找到恰当的位置后，再使用 Auto Correlation 切片，进一步分析。Merge 的输出没有 Correlation Match 这个值（即使使用了 Merge 的 Correlate 选项），也没有 Correlation 值来说明两个数据间的关系。Merge 一次只能对一张图做合并操作，而 Auto Correlation 可以一次对多张图做合并操作。

16.4 本章小结

到这里为止,完成了对 Analysis 的介绍。Analysis 是一个十分优秀的数据收集和整理维护工具,本章只是简单地介绍了 Analysis 的一些常见操作,大家可以进一步研究如何对图进行一些显示和注释的设置,加强测试报告数据的可读性和美观性。性能瓶颈的定位和分析是一个非常复杂的过程,首先需要明白每个数据的含义才能进一步发现数据间的关系,最终定位系统的短板,在修复短板后再次进行性能测试,寻找下一块短板,如此往复。

第 17 章　Web 前端性能

针对 Web 应用而言，响应时间会略微复杂一些。

Web 应用的基础是超文本传输协议(Hyper Text Transfer Protocol，HTTP)和超文本标记语言(Hyper Text Marketed Language，HTML)，HTTP 协议是一种面向连接的协议，HTML 语言则是一种用于制作超文本文档资料的简单标记语言。由于这两种协议本身具有的特性，一个 Web 界面的请求过程会更为复杂一些。

在非 Web 应用中，“请求”和“返回数据”都被当作原子操作来考虑，但是如果考虑 Web 应用本身的特点，对一个界面而言，“请求”和“返回数据”都可能是多次发生的。由于 HTTP 对浏览器下载资源的并发请求数量、cache 等方面都进行了定义和限制，以及浏览器对于 HTML 的处理过程，用户所感受到的响应时间中相当大的一部分并不完全取决于应用的后台处理所需的时间，而取决于在 Web 应用的前端(Web Frontend)。*High Performance Web Sites* 一书中指出，在 Yahoo 中，至少 50 个团队通过纯粹的前端性能相关的技巧，将最终用户的响应时间减少了 25%以上。

基于 HTTP 和 HTML 在 Web 应用中所扮演的重要角色，在讨论 Web 前端性能之前，先来对 HTML 和 HTTP 进行简单的描述。如需了解 HTML 和 HTTP 的细节，请参阅相应文档。

17.1　HTTP 协议基础理论

HTTP 用于传送 WWW 方式的数据，该协议采用了请求/响应模型。在每次交互过程中，客户端向服务器发送一个请求，请求头包括请求的方法、URI、协议版本，以及包含请求修饰符、客户信息和内容的类似于 MIME 的消息结构。服务器以一个状态行作为响应，响应的内容包括消息协议的版本、成功或者错误编码加上包含服务器信息、实体元信息以及可能的实体内容。HTTP 本身是一种非面向连接的协议，每个 HTTP 请求之间都是独立的。

通常 HTTP 消息包括客户端向服务器发送的请求消息和服务器向客户端发送的响应消息。这两种类型的消息均由一个起始行、一个或者多个头域、一个指示头域结束的空行和可选的消息体组成。

17.1.1　HTTP 协议结构

HTTP 报文由客户端到服务器的请求和从服务器到客户端的响应构成。

1. 请求报文的格式

请求报文的格式如下：

请求行　通用信息头　请求头　实体头　报文主体

请求行的格式为

```
Method[分隔符]Request-URI[分隔符]HTTP-VersionCRLF
```

说明如下：

(1) Method 表示完成 Request-URI 的方法，该字段是大小写敏感的，包括 OPTIONS、GET、HEAD、POST、PUT、DELETE 和 TRACE。方法 GET 和 HEAD 应该被所有的通用 Web 服务器支持，其他方法的实现则是可选的。GET 方法取回由 Request-URI 标识的信息，指示可以在响应时不返回消息体。POST 方法可以请求服务器接收包含在请求中的实体信息，可以用于提交表单等发送消息。

(2) [分隔符]为空格。

(3) Request-URI 遵循 URI 格式，此字段为星号(*)时，说明请求并不用于某个特定的资源地址，而是用于服务器本身。

(4) HTTP-Version 表示支持的 HTTP 版本，如 HTTP/1.1。

(5) CRLF 表示换行回车符。

HTTP 的头包括通用信息头、请求头、响应头和实体头 4 部分。每个头域由一个域名、冒号(:)和域值 3 部分组成。域名是大小写无关的；域值前可以添加任何数量的空格符。每个 HTTP 请求可以包含多个 HTTP 头域。

HTTP 报文主体则包含了 HTTP 请求的内容。对于 GET 等方法来说，报文主体为空；对于 POST 方法来说，报文主体则包含需要发送给服务器的数据。

2. 响应报文的格式

响应报文的格式为

状态行　通用信息头　响应头　实体头　报文主体

状态行由状态码和原因分析两部分构成。其中，状态码由三位数字组成，表示请求是否被理解或被满足，用于支持自动操作；原因分析是对原文的状态码作简短的描述，用来供用户使用。

响应报文中的状态码在进行 Web 应用性能测试的过程中经常遇到，说明如下。

(1) 1xx：信息响应类，表示接收到请求并且继续处理。

(2) 2xx：处理成功响应类，表示动作被成功接收、理解和接受。

(3) 3xx：重定向响应类，表示为了完成指定的动作，必须接受进一步处理。

(4) 4xx：客户端错误，表示客户请求包含语法错误或不能正确执行。

(5) 5xx：服务端错误，表示服务器不能正确执行一个正确的请求。

响应头则给出了服务器本身的一些信息，返回的HTML或其他数据内容包含在报文主体中。

17.1.2 典型的HTTP请求与响应分析

一个典型的HTTP请求如下（Firefox发出的针对images.google.com.hk的请求）：

GET http://images.google.com.hk/ HTTP/1.1
Host: images.google.com.hk
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; zh-CN; rv:1.9.2.13) Gecko/20101203 Firefox/3.6.13(.NET CLR 3.5.30729)
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*; q=0.8
Accept-Language: zh-cn,zh;q=0.5
Accept-Encoding: gzip,deflate
Accept-Chaarset: GB2312,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Cookie:
PREF=ID=f38e6b9de2b0a6a:U=e5b3ac99113a3c04:FF=1:LD=zh-CN:NW=1:TM= 1292421465:LM=1292421465:S=yzbGTkhYZ5dMh3DP;
NID=41=hEqY9NP9YEj8-1KiH9MbojYrLIqbzvrkcqHKSxNgsVAuWxxdQbg7eVKv5tecgk-M1UUAxfrYdOemvNkbkfBysi2470s2HHo13HwgmuToWUiYmAGLURAhZbz-G3poz1P

该请求的第一行指出了该请求是一个HTTP/1.1协议的请求，请求的方法是GET，请求的URI是http://images.google.com.hk。接下来的头信息包含了许多由冒号隔开的头信息对，用于指示浏览器期望接收的页面信息、浏览器本身的信息和cookie信息等。其中用粗体突出显示的是与前端性能相关的内容。

针对该请求，服务器给出的响应如下：

```
HTTP/1.1 302 OK
Location: http://images.google.com.hk/imgcat/imghp?hl=zh-CN
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Date: Sat, 18 Dec 2010 08:35:42 GMT
Server: gws
Content-Length: 246
X-XSS-Protection: 1; mode=block
```

本响应的HTTP返回码是302，告诉浏览器接下来需要跳转到另一个地址（Location指示的新地址）去获取HTML文档。状态行之下是响应头部分，与请求头一样，也包含了一些用冒号隔开的响应头信息，包括服务器类型等信息。

当浏览器根据响应的指示访问新地址时，得到的响应信息如下：

```
HTTP/1.1 200 OK
```

```
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Transfer-Encoding: chunked
Date: Sat, 18 Dec 2010 08:35:42 GMT
Expires: Sat, 18 Dec 2010 08:35:42 GMT
Cache-Control: private, max-age=0
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Server: GSE
[…HTML 文档内容…]
```

响应头信息中用粗体字标识的部分是与前端性能相关的部分。

17.1.3 与前端性能相关的头信息

HTTP 请求和响应头的某些信息是与前端性能直接相关的，下面详细介绍这些相关的头信息。

1. Accept-Encoding

Accept-Encoding 是浏览器发出的请求头中包含的头信息域之一，用于高速服务器所接收的页面文件的编码方式，如 Accept-Encoding：gzip，deflate 就是高速服务器，浏览器能够接收不压缩和使用 gzip 压缩两种方式的页面内容。页面压缩和不压缩到底能带来多大的性能差距？不妨用 ab 工具来做个简单的实验。

这次选择新浪网(www. sina. com. cn)的首页作为测试对象。

首先，要求非压缩后的页面，命令行如下：

```
ab -n 1 -c 1 http://www.sina.com.cn/
```

结果如下：

```
Server Software:        Apache/2.0.54
Server Hostname:        www.sina.com.cn
Server Port:            80

Document Path:          /
Document Length:        526 765 B

Concurrency Level       1
Time taken for tests:   5.156 25
Complete requests:      1
Failed requests:        0
Write errors:           0
Total transferred:      527 212 B
```

```
HTML transferred:        526 765 B
Requests per second:     0.20 [#/sec] (mean)
Time per request:        5015.625 [ms] (mean)
Time per request:        5015.625 [ms] (mean, across all concurrent requests)
Transfer rate:           102.48 [kB/s] received

Connection Times/ms
              min   mean  [+/-sd]  median   max
Connect:       31     31      0.0      31    31
Processing:  4984   4984      0.0    4984  4984
Waiting:       31     31      0.0      31    31
Total:       5015   5015      0.0    5015  5015
```

从结果中可以看出，获得的新浪首页 HTML 文档大小为 526 765 字节，从发出请求到接收完整 HTML 文档耗时 5.156 25s，从时间分布上来看，页面下载时间占响应时间的绝大部分。

接下来在命令行中添加 Accept-Encoding：gzip，deflate 语句，请求头告诉服务器发送压缩后的页面，命令行如下：

```
ab-n1-c1-H "Accept-Encoding:gzip,deflate"http://www.sina.com.cn/
```

结果如下：

```
Server Software:         Apache/2.0.54
Server Hostname:         www.sina.com.cn
Server Port:             80

Document Path:           /
Document Length:         117 531 B

Concurrency Level        1
Time taken for tests:    1.265 625 s
Complete requests:       1
Failed requests:         0
Write errors:            0
Total transferred:       118 002 B
HTML transferred:        117 531 B
Requests per second:     0.79 [#/sec] (mean)
Time per request:        12 653 625 [ms] (mean)
Time per request:        12 653 625 [ms] (mean, across all concurrent requests)
Transfer rate:           90.86 [kB/s] received

Connection Times/ms
              min     mean    [+/-sd]    median    max
Connect:      31      31      0.0        31        31
```

```
Processing:  12 341 234    0.0        12 341 234
Waiting:         31     31   0.0         31         31
Total:         1265   1265   0.0       1265       1265
```

从结果可以看到，响应时间变成了 1.265 625s，减少的时间即是下载所需时间，虽然实际上后一次网络下载速度小于前一次，但由于页面大小从 526 765 字节减小到了 117 531 字节，因此下载所需要的时间从 5s 多减少到了 1.2s。从这里可以看出，服务器端对每一个页面为支持 gzip 压缩的浏览器生成压缩后的页面多么重要。

2. Connection

HTTP 是一种非面向连接的、无状态的协议。每一个 HTTP 请求与其他 HTTP 请求都是完全独立的，因此从理论上来说，每一个 HTTP 请求都会经历一个“建立连接——请求页面或资源——获得页面或资源——断开连接”的过程。但建立和断开连接需要一定的时间和资源开销，对于非常小的页面和资源文件来说，很可能建立连接所消耗的时间甚至大于页面或资源的处理时间。为了让响应时间尽可能缩短，HTTP 引入了持久连接。如果浏览器与服务器约定了持久连接，当某个资源文件或是 HTML 文档传输完成后，连接并不立即关闭，而是等待一段时间，在这段时间内如果需要传输其他资源文件或 HTML 文档，则复用该连接；只有在一段时间内没有任何传输需要发生时，才关闭该连接。

用于控制持久连接的 HTTP 请求头就是 Connection。当 Connection 的值设定为 keep-alive 时，浏览器与服务器之间约定使用持久连接。

3. Expires

HTTP 响应数据头中包含一个 Expires 域，该域的值用于返回数据的到期时间。例如，17.1.2 节中给出的 HTTP 响应头中包含 Expires：Sat，18 Dec 2010 08：35：42GMT 信息，这就是服务器告诉浏览器，返回的 HTML 文件内容到 2010 年 12 月 18 日 8 点 35 分 42 秒到期(GMT 时间)。

在日常使用浏览器时，通常都会感觉某个页面在第一次打开时的速度特别慢，这是因为浏览器的缓存机制在起作用。但是缓存机制究竟是如何工作的呢？浏览器怎么知道什么时候该使用缓存中的数据，什么时候该从服务器上重新取数据呢？Expires 头给出的信息就是一句：当当前时间小于 Expires 指定的时间时，浏览器从缓存中直接获取响应的资源文件或 HTML 文档；而当当前时间大于 Expires 指定的时间时，浏览器向服务器的发送请求获取该资源。

17.1.2 节中给出的 HTTP 响应示例中，响应头中 Date 和 Expires 的内容相同，都是 Sat，18 Dec 2010 08：35：42GMT，Date 指示的是请求返回时的当前时间，Expires 设置为与 Date 时间一样，不是不能使用缓存了吗？的确是这样的，由于某些原因，Google 希望浏览器每次重新获取 http://images.google.com.hk/imgcat/imghp？hl=zh-CN 页面的内容。而对于页面上的图片，在第一次访问页面时，其中某图片文件的 HTTP 相应内容如下：

```
Content-Type: image/jpeg
Last-Modified: Tue, 15 Sep 2009 22:22:42 GMT
Date: Sat, 18 Dec 2010 13:09:32 GMT
Expires: Sun, 18 Dec 2011 14:03:20 GMT
X-Content-Type-Options:nosniff
Server: sffe
Content-Length: 4225
X-XSS-Protection: 1; mode=block
Cache-Control: public, max-age=31536000
```

可以看到，对该图片设置的 Expires 时间为返回页面时间的 1 年之后。因此，在第二次访问 images.google.com.hk 首页时，所有的图片文件都是直接从缓存中获取的。

17.2 浏览器访问 URL 原理

不同浏览器的工作方式不完全一样，大体上，浏览器的核心是浏览器引擎(Browser Engine)，目前市场占有率最高的几种浏览器几乎都使用了不同的浏览器引擎：IE 使用的是 Trident，Firefox 使用的是 Gecko、Safari 和 Chrome 使用的则是 Webkit。不同浏览器引擎对 W3C 规范的支持不尽相同，在具体功能的实现上也不完全一致。因此，除了一些特别限制的区别外，本节不会着重讨论各个浏览器引擎之间的差异，而是主要描述浏览器从输入 URL 地址开始到页面完全可用的大致过程。

17.2.1 连接到 URL 服务器

用户在地址栏中输入一个 URL，按 Enter 键后浏览器打开该 URL，浏览器做的第一件事就是寻找该 URL 所在网站的 IP 地址。然后，浏览器向该地址发起连接请求，建立到服务器的连接。

17.2.2 获取页面对应的 HTML 文档

当建立连接后，浏览器向服务器发送 HTTP 请求，请求 URL 对应的 HTML 文档。不管请求的 URL 是一个静态的 HTML 文件，还是一个动态脚本(ASPX、PHP 或 JSP)，服务器返回给浏览器的一定是一个 HTML 文档。该 HTML 文档就是浏览器需要呈现的页面。

17.2.3 解析文档并获取所需要的资源

浏览器在获取 HTML 文档后会对文档进行解析，目的是知道该页面需要哪些资源以及生成 DOM 树。生成 DOM 树的工作与下载页面上需要的其他资源同时进行。大致来说，浏览器会逐行分析 HTML 文档，一旦发现一个标签，就会根据标签的要求分配对

指定资源的下载。当 DOM 树生成后，DOMContentLoaded 事件被触发（IE 没有实现 DOMContentLoaded 事件）。

比较有趣的是浏览器下载资源的过程。首先，理论上浏览器并行下载页面需要的所有资源会带来最好的性能体验，但由于服务器要保证对尽可能多的用户的支持，因此 HTTP/1.1 规定了每个客户端（浏览器）只能与每个服务器建立两个连接。在 HTTP/1.1 诞生的年代，由于服务器处理能力的限制，这个限制自然是合理的。但随着互联网技术的发展，服务器处理能力的提高以及负载均衡的大量应用，该限制就显得不那么合理了。因此，在较新版本的浏览器中，大多没有严格遵循该限制，如 Firefox 3.6.13 中文版默认将每个服务器的连接数限制为 6 个。但不管具体的数量如何，该并发的限制确实是存在的。

其次，并非所有的元件都可以被并行下载。一般情况下，页面中包含两类需要被执行的 JavaScript 脚本，一类是直接用<script>标签标识的内嵌 JS 语句；另一类则是引用的外部 JS 文件。由于浏览器需要按照 JS 遵循其在 HTML 文档中的顺序来执行 JS，在较早的浏览器（如 IE 6 和 IE 7，Firefox 3.5 以前的版本）中，当 JS 文件下载时，页面上其他所有资源的下载均会被阻塞，只有当 JS 文件下载并执行完成后，浏览器才继续下载页面上的其他资源。

17.2.4 onload 事件

当 HTML 文档解析完成，生成了 DOM 树，所有页面需要的资源文件都已经成功下载和执行（对于 JavaScript 文件）后，浏览器会发出 onload 事件并回调 HTML 文档中的 onload 函数。

对浏览器来说，onload 事件是最接近页面就绪的时间，因此，许多前端性能工具都会把 onload 事件作为一个显著的标识。在 Firebug 给出的结果中，红色线条指示的就是 onload 事件被触发的时间点。在 HttpWatch 工具中，同样用红色的线条标识 onload 事件得到发生。

但是，是不是 onload 事件执行完成就标志着页面已经就绪，用户可以在其上进行操作了？并非如此。因为在 onload 执行完成后，页面上可能还有一些 JavaScript 脚本需要在其后运行。在大量应用 Ajax 技术的网页上，onload 执行完成后页面仍然没有完全就绪的情况非常常见，例如，当 onload 执行完成后利用页面上的 JavaScript 发送 Ajax 请求，拿到请求的返回结果后再修改某些 DOM 元素，以达成用户观感上的提速。

17.3 如何提高 Web 前端的性能

在 17.2 节中，详细分析了浏览器打开 URL 的过程。从该过程的描述中可以看到，要提升前端的性能，有两大思路：

(1) 减少页面加载需要的时间。

(2) 提升用户的观感，让用户觉得页面更快（对于纯粹的数据展示页面，可以通过让

页面尽快开始显示达到这样的效果)。

17.3.1 减少网络时间

浏览器从服务器获取 HTML 文档和资源都需要经历“DNS 解析——建立连接——获取内容——断开连接”的过程。如果能够减少 DNS 解析和文件在网络上的传输时间,系统性能自然会得到提升。

1. 使用 DNS 缓存技术

使用 DNS 缓存技术能够让用户获得更快的 DNS 解析时间,当然,一般而言,由于浏览器本身有一定的 DNS 缓存机制(不同浏览器有不同的实现),通常情况下,服务端的 DNS 缓存并不能带来太大的性能提升。

2. 减小需要传输文件的尺寸

减少需要传输文件的尺寸是一个提升性能的方法。在网络带宽有限的情况下,这种方法能带来巨大的好处。通过使用 gzip 压缩页面,能够有效提升页面速度。除了确保使用 gzip 压缩页面外,使用混淆等方法尽量减小 JS 文件和样式表的大小,从 JS 文件盒样式表中去除不需要使用的部分等,都可以起到减小需要传输文件尺寸的作用。

3. 加快文件传输速度

Internet 网站的用户通常会分布在一个较广阔的区域中。Internet 本身的多层次网络结构导致了从某些节点到另一些节点之间的可用带宽和网络传输速度都比较慢,在这种情况下,使用 CDN 技术,让用户尽可能访问到对用户节点而言更快速的服务器就可以加快文件传输速度。以中国的状况为例,联通、电信和移动三大运营商之间并没有建立很好的互连互通,因此在网络用户广泛分布在这三个运营商网络的情况下,通常需要在三个运营商所在的网络中设置 GDN 服务器。另外,由于地域原因建立 CDN 也是一种常见的做法。

17.3.2 减少发送请求的数量

减少需要发送的请求数量显然可以提升性能。在短连接情况下,每一个请求都需要经过“建立连接——发送数据——断开连接”的过程,因此减少必需的请求数量可以带来显著的性能提升。即使使用持久连续方式,由于浏览器与每个服务器之间建立的持久连接数量是有限的,减少必需的请求数量也能给性能提升带来帮助。

1. 利用浏览器缓存

充分利用浏览器的缓存机制可以有效提升系统性能。在 17.1.3 节中已经讨论了浏览器缓存的具体实现,为了充分利用浏览器缓存机制,需要在服务端保证每个可以被缓存

的资源在被服务端返回时附带合适的 Expires 头信息。此外，为了保证尽可能多的内容可以被缓存，也就要求网站尽可能将页面中较少改变的部分提取出来。归纳而言，以下措施有助于充分利用缓存：

(1) 保证服务端返回资源的响应头带有 Expires 信息，使得资源可以被缓存。

(2) 用引用方式使用样式表和 JS 脚本。如果使用内嵌的样式表和 JS 脚本，每次 HTML 文档的变化都会导致样式表和 JS 脚本需要重新加载，无法充分利用缓存。当然，在没有缓存或样式表与 JS 脚本经常变动的情况下，用引用方式使用样式表和 JS 脚本反而会导致更多的 HTTP 请求。

(3) 使更多的 URI 可以被浏览器缓存。许多网站使用脚本生成需要返回的图片或 JS 资源文件，而该脚本所在的 URI 又附带了一些经常变化的参数，这就导致了这些内容无法被缓存(缓存要求 URI 严格一致)。例如，如果某网站页面上的图片 URI 为 http://www.somedomain.com/getpi?category=[categoryid]&pos=[position]，若给定图片每次的 URI 不同，即使两个不同的 URI 实际内容是同一张图片，浏览器也无法使用已有的缓存图片。

2. 使用合并的图片文件

当页面上包含许多小图片文件时，可以考虑将小图片文件合并成一个大图片文件，在页面上使用 CSS Sprites 技术将大图片显示为分隔开的小图片。在没有缓存的情况下，将许多小图片文件合并成大图片文件可以大量减少 HTTP 请求数。当然，选择将哪些小图片合并成大图片也需要综合考虑，如果某些小图片经常变化，而另一些相对稳定，则不宜将所有小图片合并成大图片，因为这样会导致大图片经常改变，从而无法利用浏览器的缓存机制。

17.3.3 提高浏览器下载的并发度

最简单的提高浏览器下载并发度的方法是在浏览器上进行设置，允许浏览器与每个服务器建立不限数量的连接。但如果浏览器与某个服务器建立了过多的连接，很可能会被服务器判定为攻击而将其拉入黑名单；即使服务器运行这类操作，也需要用户在自己的浏览器上进行设置，因此并不是一个好办法。

更好的方式是充分利用浏览器的特性，在无须修改任何浏览器设置的情况下，达到更好的下载并发度。

1. JS 文件放在 HTML 文件的最后

在某些浏览器(如 IE 6)上，JS 文件的下载和执行会阻止其他页面资源文件的下载和执行，直到 JS 文件下载和执行完成后，其他资源文件才可以开始下载和执行。因此，将 JS 文件放在 HTML 文档的最后可以保证 JS 文件不会组织任何其他元素的下载。当然，在 IE 6 中，也可以使用一些特别的技巧，例如使用 document.write 方法在文档中写入 JavaScript 脚本，或是将 JavaScript 脚本以元素处理，使用 appendChild 方法加入文档，达

成浏览器并行下载 JS 文件的效果，这些技巧对于提升前端通用性有所帮助，这需要程序员仔细考虑对页面上的 JS 文件的划分，然后使用这些技巧达成对性能的提升。关于让 JS 文件并行下载的讨论在互联网上有很多，例如文章 http://blog.rakeshpai.me/2009/03/downloading-javascript-files-in.html 就详细介绍了这种技巧的实现。

2. 使用多个域名

浏览器对服务器的连接限制是基于域名的。例如，如果某个服务器 S 拥有两个域名 a.com 和 b.com，在浏览器限定最多与同一个域名建立两个连接的情况下，浏览器实际上可以与服务器 S 建立 4 个连接。相信大家在访问一些大型网站时会注意到，大型网站往往拥有几个域名，例如，用 image.xx.com 存放图片文件等静态资源文件；用 stat.xx.com 放置用于统计的 JS 脚本等。这样可以根据不同类型的资源（静态或动态），选择最合适的服务器进行部署。例如，Nginxz 在处理静态文件上有绝对优势，那就将所有的静态资源文件设置在一台 Ngins 服务器上，而将脚本放在其他服务器上。另一方面，将页面放置在相同的服务器上也可以提高下载的并发度，从而提高系统性能。

17.3.4 让页面尽早开始显示

在 17.3.1 节中，提到了 JS 文件和样式表在不同浏览器上对性能有不同的影响。Steve Souders 的一篇名为 *Frontend SPOP* 的文章中给出了一个样式表、JS 脚本以及 @font-face 类型对页面渲染的影响，其部分结果如表 17-1 所示。

表 17-1 不同浏览器下 JS、样式表和@font-face 的影响

前端 SPOF 测试	Chrome	Firefox	IE	Opera	Safari
外部脚本（JS 脚本）	下部空白	下部空白	下部空白	下部空白	下部空白
样式表	下部空白	闪烁	下部空白	闪烁	下部空白
内嵌的@font-face	延迟	闪烁	闪烁	闪烁	延迟
带@font-face 的样式表	延迟	闪烁	完全空白	闪烁	延迟
带@font-face 的脚本	延迟	闪烁	完全空白	闪烁	延迟

表中各结果详细描述如下：

（1）下部空白指当前元素以上的所有其他元素都在屏幕上被显示出来。

（2）闪烁指 DOM 元素一旦下载完成就会被显示出来，但样式表或自体下载完成后会导致页面重画，产生闪烁。

（3）延迟指使用@font-face 类型的文本直到字体下载完成才开始显示。

（4）完全空白指页面上什么都不显示，只有一片空白。

虽然不同的浏览器有不同的处理方式，但从它们的行为中可以总结出几个规律：

（1）当开始下载和执行 JS 文件时，页面停止显示新的元素。

（2）当载入样式表时，页面上要么停止显示新的元素，要么出现闪烁。

(3) IE 浏览器在使用带@font-face 类型的样式表和脚本时,用户体验较差。

因此,根据表 17-1 中的数据,对所有浏览器都有效的测量是:

(1) 将样式表的引用放在 HTML 文档的开头(如放在<Head>标签中)。这样可以让样式表在一开始就被下载下来。一旦样式表下载完成,浏览器就可以使用样式表中定义的样式开始在屏幕上显示页面元素。另外,也避免了新样式可能带来的屏幕显示的重绘。

(2) 将 JS 的引用放在 HTML 文档的最后。这样 JS 文件的下载和执行会在所有页面下载完成之后,不会阻止其他页面元素的显示,从用户观感上来说,JS 文件的下载和执行时间完全不会被用户感觉到。

17.3.5 其他

前面主要讨论例如提升前端性能的一些通用法则。除了这些法则之外,还有更多更细致的针对 JS 或样式表的具体技巧。2010 年 Web 性能与运维大会(Velocity China 2010)在北京召开,会上多位业内重量级人士发表了面向前端性能的演讲并明确指出:前端性能是目前 Web 应用性能的一个主要方向,前端性能优化能够带来极大的网站性能提升。当然,不好的前端性能对于网站的打击也是致命的。

17.4 单机前端性能工具介绍

本章首先简单描述了 HTTP,阐述了浏览器打开 URL 的过程,并给出了提高前端性能的一些具体方法。但是,如何才能对一个已有系统的前端性能状况进行性能评估呢?本节就围绕单机前端性能评估工具进行展开,这些工具有:Firebug、HttpWatch、PageSpeed 和 DynaTrace。

17.4.1 Firebug 工具

Firebug 工具是一个备受推崇的、强大的 Web 开发工具。

(1) 它提供了方便的查看页面元素功能,允许用户以鼠标指示、DOM 树等方式查看任意界面元素;

(2) 提供了 JavaScript 控制台,允许用户在控制台直接调试 JavaScript;

(3) 提供了可视化的 CSS 标尺,方便用户调整页面布局;

(4) 提供了网络面板,允许用户获取每个页面被加载过程中的页面元素下载和执行效果。

除此之外,Firebug 工具同时还提供了非常好的扩展,不少好用的工具就是基于其扩展而建立的,如 YSlow 和 Page Speed 工具。

接下来的篇幅中,人们的关注点会集中在 Firebug 的网络面板上,探究如何通过网络面板评估前端性能。

Firebug 以 Firefox 的插件形式存在，一旦在 Firefox 上安装 Firebug，Firefox 的“工具”菜单下会多出一个 Firebug 的菜单项，以下以 Firebug 2.0.3 和 Firefox 3.1.0 为例，说明 Firebug 网络面板的使用方法。安装完 Firebug 后，单击菜单栏中的“工具”→Web 开发者→Firebug，打开 Firebug，或者直接按 F12，打开 Firebug 后，其主窗口默认位于浏览器的下半部分，如图 17-1 所示。



图 17-1　Firebug 主窗口

单击主窗口中的“网络”标签，选择“启用”网络面板，即可在 Firefox 中正常使用该网络面板。启用网络面板后，在 Firefox 的地址栏中输入 URL 并打开，就会在该网络面板上显示本次请求处理的网络细节。

图 17-2 是在 Firefox 中打开 www.baidu.com 后在网络面板中显示的结果。

图 17-2　打开 www.baidu.com 显示在网络面板中的内容

网络面板展示了该页面在浏览器处理过程中的所有网络交互、每一个请求的返回值、返回内容的大小以及相应的时间线。通过单击网络面板上的按钮还可以按界面元素的类型分类显示元素的下载情况。“保持”按钮用于保持已有的显示结果，单击“保持”按钮后，刷新页面仍然会保留已记录的请求。

对于每一个请求，单击左侧的加号，可以看到该请求的请求头、响应头和是否使用缓存等信息。网络面板中的蓝色(左侧)和红色(右侧)线条分别表示 DOMContentLoaded 事件发生和 onload 事件触发的时间点。当 DOM 树生成后，DOMContentLoaded 事件立即被触发，蓝色线条指示的时间点就是该事件发生的时间点；当所有页面元素下载完成后，onload 事件被触发，HTML 文档中的 onload 函数被执行，红色线条指示的是 onload 事件被触发的时间点。然而，无论是 DOMContentLoaded 消息，还是 onload 消息，都不意味着页面已经完全就绪。如果关心用户什么时候能看到页面内容开始渲染，则可以在网络面板中选择“显示 Paint 事件”命令。单击“网络”标签右侧的下拉按钮，在菜单中选

择"显示 Paint 事件"命令，即可看到该页面上发生的每一次 Paint 事件的时间点。一个 Paint 事件意味着页面上的一次绘制。

选择"显示 Paint 事件"命令后，再次访问 www.baidu.com，结果如图 17-3 所示。

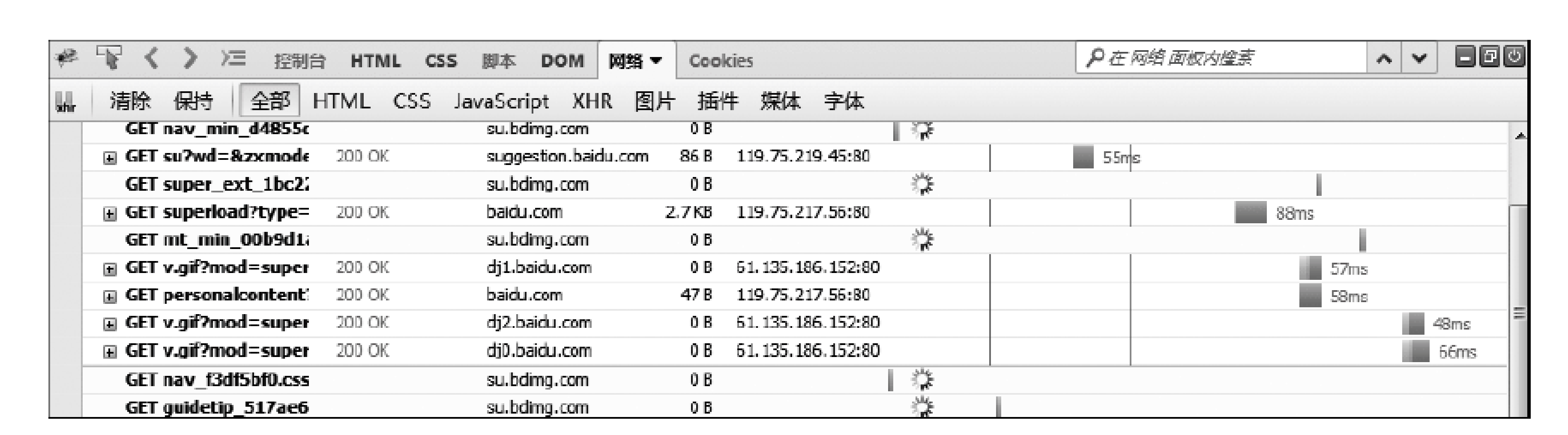


图 17-3　选择"显示 Paint 事件"命令后的结果

17.4.2　HttpWatch 工具

HttpWatch 工具是可在 IE 和 Firefox 下使用的网页数据分析工具。与 Firebug 不同，HttpWatch 是一个商业工具，其提供了一个 Basic 的免费版本。对于前端性能而言，Basic 免费版本基本够用。

在 Windows 平台上安装 HttpWatch 工具非常简单，下载安装包后直接执行即可。安装完成后，在 IE 的"工具"菜单下会多出一个 HttpWatch 菜单项，选择该项目就可以在浏览器中打开 HttpWatch 的主窗口。以下以 IE 10 和 HttpWatch Basic 9.3 为例，说明该工具的使用。如图 17-4 所示是 HttpWatch 工具的主窗口。要开始使用该工具，首先需要单击窗口左上角的 Record 按钮，然后在 IE 地址栏中输入需要访问的 URL。如图 17-5 所示是访问 www.baidu.com 得到的结果。建议在页面加载完成后，单击 Stop 按钮停止录制，否则，只要 IE 有任何发出的请求，HttpWatch 就会将其记录下来。



图 17-4　HttpWatch Basic 的主窗口

与 Firebug 类似，HttpWatch 工具给出了浏览器请求该 URL 过程中发生的全部 HTTP 交互，列出了每一个元素的 HTTP 返回码、元素大小以及按照时间序列给出的页面元素下载时间等信息。此外 HttpWatch 在 Page Event 选项卡中还给出了 Render Start、Page Load 和 HTTP Load 三个时间。其中，Render Start 描述的是浏览器开始渲

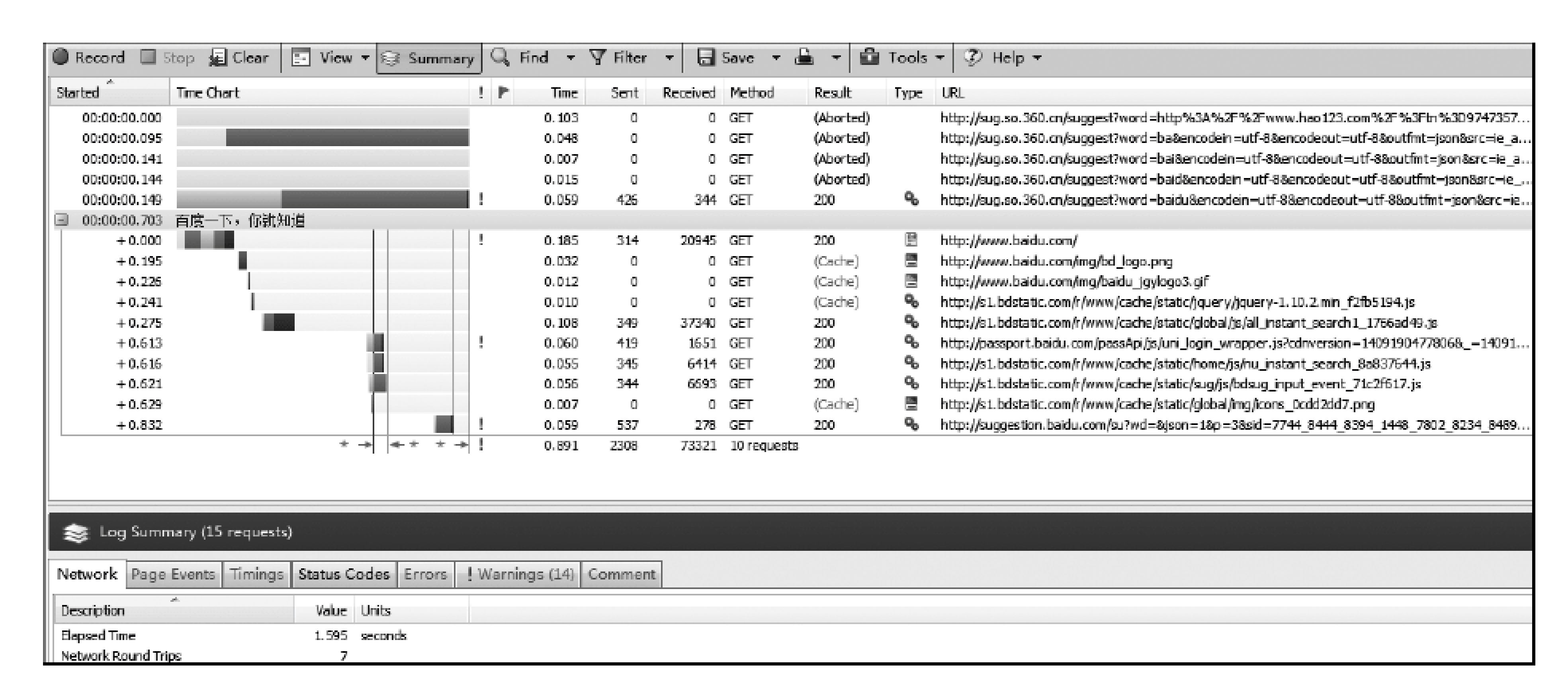


图 17-5 打开 www. baidu. com 后的 HttpWatch 主窗口

染页面的时间；Page Load 描述的是 onload 事件触发的时间；HTTP Load 描述的是页面上最后一个请求发送和接收完毕的时间点。这三个时间分别对应 Firebug 中的第一个 Paint 事件发生时间、onload 时间以及收到最后一个 HTTP 响应的结束时间。

17.4.3 Chrome 自带的开发工具

Chrome 是 Google 公司推出的一款浏览器产品，该浏览器的主要特点是快速、安全、简介。除此之外，Chrome 浏览器从创建之初便自带了许多方便开发工程师调试的工具。

打开 Chrome 浏览器，选择“工具”→“开发者工具”命令，在打开的窗口中单击 Resources 标签，允许对该 Session 中的资源进行跟踪，在浏览器中输入 URL 并访问。仍然以访问 www. baidu. com 为例，在主窗口中切换到 Timeline 界面，单击左上角的圆形图标开始录制，就能在工具中看到录制时间段内所有浏览器产生的事件。从图 17-6 中可以清楚地看到请求发送的时间点、浏览器每个事件发生的时间点以及浏览器开始渲染的时间点；也能看到每次重绘的 Paint 事件，甚至每次重绘的屏幕区域都在工具中进行了标识。

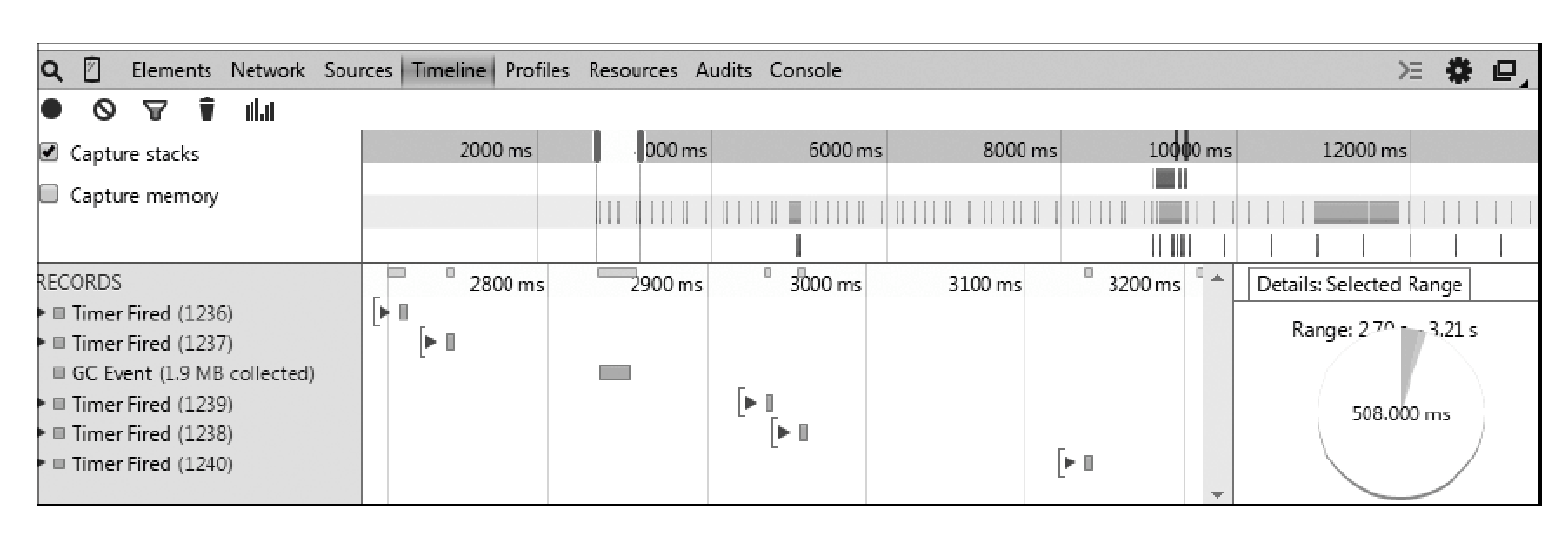


图 17-6 Chrome 自带开发工具的 Timeline 界面

17.4.4 Page Speed 工具

Page Speed 工具是一个基于 Firebug 工具的前端性能优化工具，该工具由 Google 公司创建并发布。PageSpeed 工具依据一些规则对页面进行检查，查找可以优化的地方并给出相应的建议。可以从网站 https://developers.google.com/speed/pagespeed/insights_extensions?hl=en 上下载安装 Page Speed 插件，但安装 Page Speed 插件前，需要首先安装 Firebug 插件。安装 Page Speed 插件后，打开 Firebug 主窗口，单击 Page Speed 选项，打开 Page 界面，如图 17-7 所示。

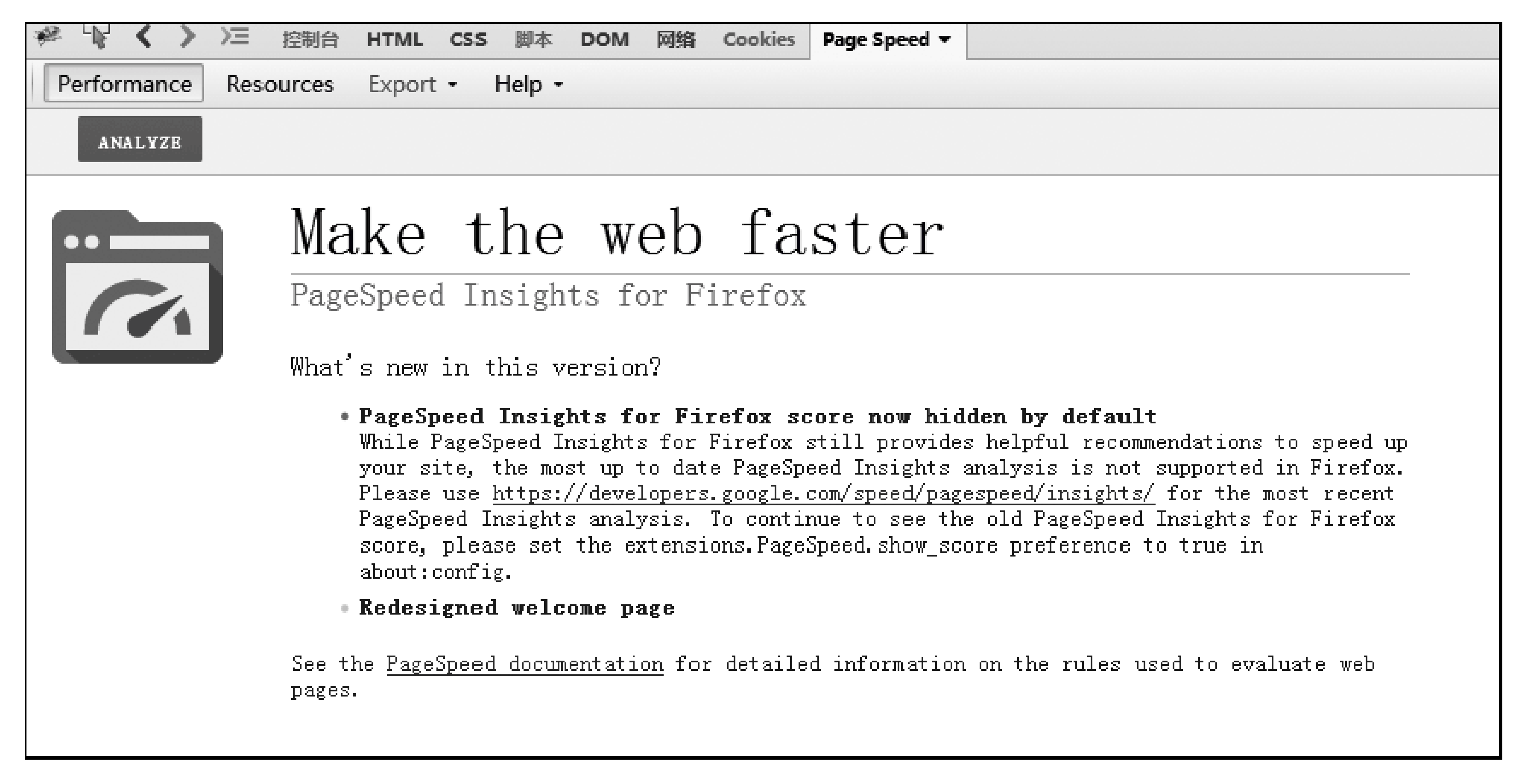


图 17-7 打开的 Page Speed 界面

使用浏览器加载一个页面，然后单击 Page Speed 界面中的 Performance 按钮，Page Speed 会根据一些预先定义的规则检查该页面，提供可以提高前端性能的建议。如图 17-8 所示，Page Speed 给出了一个对页面前端性能的评分，并详细列出了每一个检查项目的结果以及可能的改进建议。单击每个条目左侧的加号图标可以展开具体的条目。



图 17-8 对某页面检查的结果

Page Speed 提供的建议是多方面的，前端性能的方法基本都已经包含在 Page Speed

的检查内容中，除此之外，Page Speed 还包含了许多其他方面的最佳实践和建议。完整的 Page Speed 的规则列表请参见 https://developers.google.com/speed/docs/insights/rules。

17.4.5 DynaTrace AJAX Edition 工具

DynaTrace AJAX Edition 工具是 Windows 平台上的免费工具，该工具提供了非常强大的前端性能测试支持。从该工具的名称就可以看出，其主要是针对大量使用 Ajax 技术的应用开发的。可以从 http://ajax.dynatrace.com/ajax/en/获得 DynaTrace AJAX Edition 工具，此处使用的版本为 4.2。安装该工具后，第一次打开时系统会提示用户输入 Community Account，如图 17-9 所示，如果用户有 Community Account，就可以充分利用 DynaTrace 的平台，与其他人交流前端性能测试经验，以及从 DynaTrace 网站获得一些可供比较的基准结果。



图 17-9 新建 DynaTrace AJAX Edition

打开该工具后，单击工具栏最左侧按钮旁的下拉按钮，从下拉菜单中选择 New Run Configuration 命令，输入项目名称和要访问的网站 URL，就可以开始执行任务了。

DynaTrace AJAX Edition 工具会打开一个 IE 窗口并访问指定的 URL。打开 IE 窗口后，在 DynaTraceAJAX Edition 的主窗口中会显示对该网页的总评分以及各个分项目的评分信息，如图 17-10 所示。

DynaTrace AJAX Edition 能够给出浏览器在访问给定 URL 时的许多信息，如页面中各元素的下载和执行时间点、页面对浏览器缓存的使用情况等，并能够根据一些预定义的规则给出建议。

除此之外，DynaTrace AJAX Edition 工具最突出的功能是对页面上的 JavaScript 调

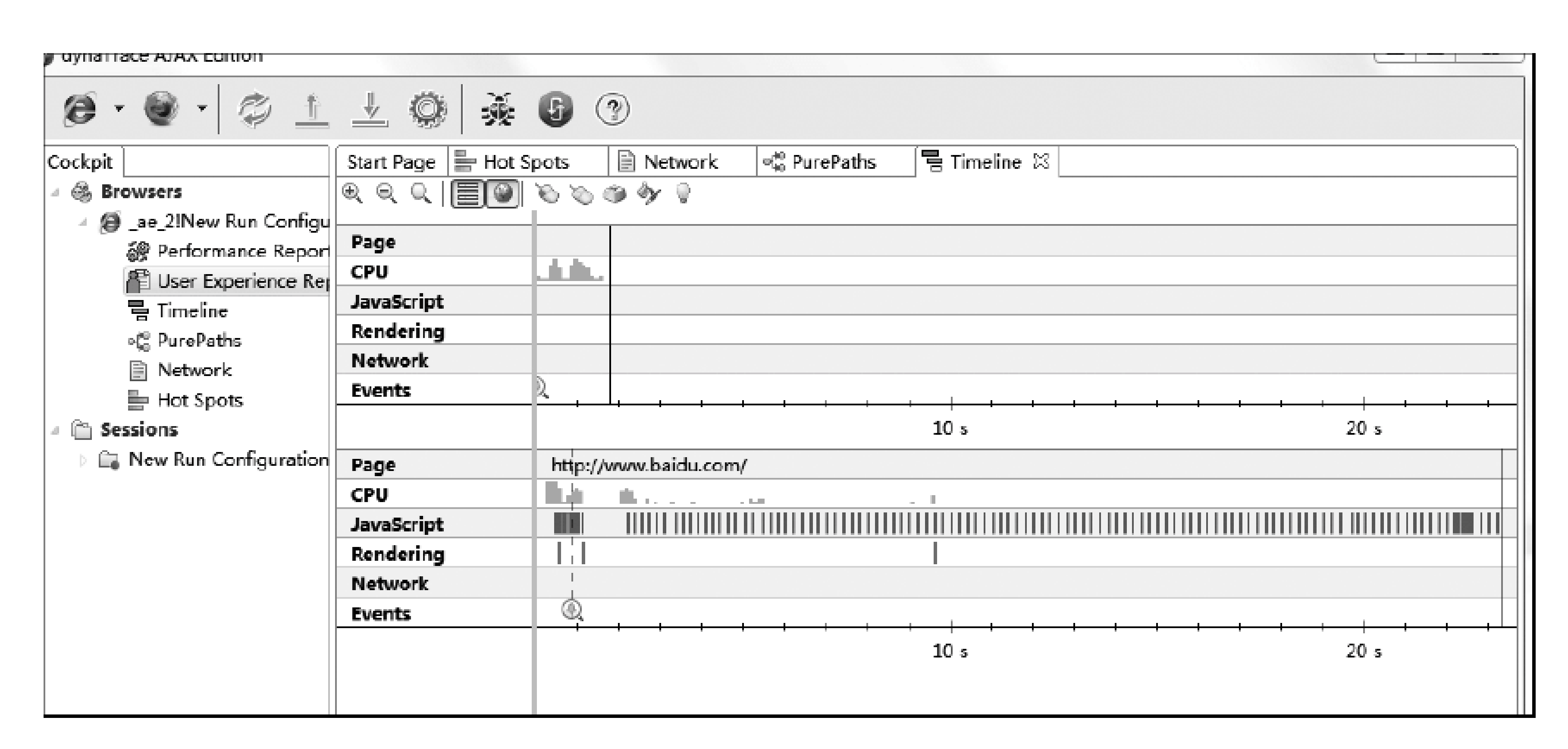


图 17-10　DynaTrace AJAX Edition 对某网站的检查结果

优。如图 17-11 所示的是 DynaTrace AJAX Edition 对某网站的分析结果，图中展示了页面上所有 JavaScript 脚本执行的时间，单击脚本列表，可以看到脚本函数的执行顺序以及每个脚本的具体内容。

Start Page | Hot Spots | Network | PurePaths | Timeline | Network

<session>

Showing: all

Time Chart	Start Time...	Duration [...	JS [ms]	CPU Time...	Size	Activity ki...	Details
	10.33	34.40	22.50	15.60	204	JavaScript	<script>
	10.37	117.66	55.93	46.80	1004	JavaScript	<script>
	10.51	272.97	26.60	31.20	1637	JavaScript	<script>
	10.54	26.73	24.85	0.00	10	JavaScript	<script>
	10.57	0.03	0.03	0.00	4	JavaScript	readystatechange event on <sc
	10.57	0.02	0.02	0.00	4	JavaScript	readystatechange event on <sc

Activity [not aggregated]	Start [ms]	Duration [m...	API
JavaScript Execution	0.00	0.02	Java
n.onreadystatechange(null, *)	0.00	0.01	<htn
readyState	0.01	0.00	DON
readyState	0.01	0.00	DON

图 17-11　DynaTrace AJAX Edition 对某网站的检查结果

DynaTrace AJAX Edition 工具的功能非常多，这里只是简单介绍了该工具的基本使用方式，更详细的使用方式请读者自行探索。

在网站页面性能优化的层面，为读者列举雅虎团队的经验。

17.5　雅虎团队经验：网站页面性能优化的 34 条黄金守则

17.5.1　尽量减少 HTTP 请求次数

终端用户响应的时间中，有 80%用于下载各项内容。这部分时间包括下载页面中的

图像、样式表、脚本、Flash 等。通过减少页面中的元素可以减少 HTTP 请求的次数，这是提高网页速度的关键步骤。

减少页面组件的方法其实就是简化页面设计。那么有没有一种方法既能保持页面内容的丰富性又能达到加快响应时间的目的呢？这里有几条减少 HTTP 请求次数同时又保持页面内容丰富的技术。

合并文件是通过把所有的脚本放到一个文件中来减少 HTTP 请求的方法，如可以简单地把所有的 CSS 文件都放入一个样式表中。当脚本或者样式表在不同页面中使用时需要做不同的修改，这可能会相对麻烦点，但即便如此也要把这个方法作为改善页面性能的重要一步。

CSS Sprites 是减少图像请求的有效方法。把所有的背景图像都放到一个图片文件中，然后通过 CSS 的 background-image 和 background-position 属性来显示图片的不同部分。

图片地图是把多张图片整合到一张图片中。虽然文件的总体大小不会改变，但是可以减少 HTTP 请求次数。图片地图只有在图片的所有组成部分在页面中是紧挨在一起的时候才能使用，如导航栏。确定图片的坐标和可能会比较烦琐且容易出错，同时使用图片地图导航也不具有可读性，因此不推荐这种方法。

内联图像是使用 data：URL scheme 的方法把图像数据加载页面中。这可能会增加页面的大小。把内联图像放到样式表（可缓存）中可以减少 HTTP 请求同时又避免增加页面文件的大小。但是内联图像现在还没有得到主流浏览器的支持。

减少页面的 HTTP 请求次数是首先要做的一步。这是改进首次访问用户等待时间最重要的方法。如同 Tenni Theurer 的他的博客 *Browser Cache Usage-Exposed* 中所说，HTTP 请求在无缓存情况下占去了 40%～60%的响应时间。让那些初次访问你网站的人获得更加快速的体验吧！

17.5.2 减少 DNS 查找次数

域名系统（DNS）提供了域名和 IP 的对应关系，就像电话本中人名和他们的电话号码的关系一样。当用户在浏览器地址栏中输入 www.dudo.org 时，DNS 解析服务器就会返回这个域名对应的 IP 地址。DNS 的解析过程同样也是需要时间的。一般情况下返回给定域名对应的 IP 地址会花费 20～120ms 的时间，而且在这个过程中浏览器什么都不会做直到 DNS 查找完毕。

缓存 DNS 查找可以改善页面性能。这种缓存需要一个特定的缓存服务器，这种服务器一般属于用户的 ISP 提供商或者本地局域网控制，但是它同样会在用户使用的计算机上产生缓存。DNS 信息会保留在操作系统的 DNS 缓存中（微软 Windows 系统中的 DNS Client Service）。大多数浏览器有独立于操作系统以外的自己的缓存。由于浏览器有自己的缓存记录，因此在一次请求中它不会受到操作系统的影响。

Internet Explorer 默认情况下对 DNS 查找记录的缓存时间为 30min，它在注册表中的键值为 DNSCacheTimeout。Firefox 对 DNS 的查找记录缓存时间为 1min，它在配置文

件中的选项为 network.dnsCacheExpiration(Fasterfox 把这个选项改为了 1h)。

当客户端中的 DNS 缓存都为空时(浏览器和操作系统都为空),DNS 查找的次数和页面中主机名的数量相同,这其中包括页面中 URL、图片、脚本文件、样式表、Flash 对象等包含的主机名。减少主机名的数量可以减少 DNS 查找次数。

减少主机名的数量还可以减少页面中并行下载的数量。减少 DNS 查找次数可以节省响应时间,但是减少并行下载却会增加响应时间。笔者的指导原则是把这些页面中的内容分割成至少两部分但不超过四部分。这种结果就是在减少 DNS 查找次数和保持较高程度并行下载两者之间的权衡了。

17.5.3 避免跳转

跳转是使用 301 和 302 代码实现的。下面是一个响应代码为 301 的 HTTP 头:

```
HTTP/1.1 301 Moved Permanently
Location:http://example.com/newuri
Content-Type: text/html
```

浏览器会把用户指向 Location 中指定的 URL。头文件中的所有信息在一次跳转中都是必需的,内容部分可以为空。不管它们的名称,301 和 302 响应都不会被缓存除非增加一个额外的头选项,如 Expires 或者 Cache-Control 来指定它缓存。<meat />元素的刷新标签和 JavaScript 也可以实现 URL 的跳转,但是如果必须要跳转,最好的方法就是使用标准的 3XXHTTP 状态代码,这主要是为了确保"后退"按钮可以正确地使用。

但是要记住跳转会降低用户体验。在用户和 HTML 文档中间增加一个跳转,会拖延页面中所有元素的显示,因为在 HTML 文件被加载前任何文件(图像、Flash 等)都不会被下载。

有一种经常被网页开发者忽略却往往十分浪费响应时间的跳转现象。这种现象发生在当 URL 本该有斜杠(/)却被忽略掉时。例如,当要访问 http://astrology.yahoo.com/astrology 时,实际上返回的是一个包含 301 代码的跳转,它指向的是 http://astrology.yahoo.com/astrology/(注意末尾的斜杠)。在 Apache 服务器中可以使用 Alias 或者 mod_rewrite 或者 the DirectorySlash 来避免。

连接新网站和旧网站是跳转功能经常被用到的另一种情况。这种情况下往往要连接网站的不同内容然后根据用户的不同类型(如浏览器类型、用户账号所属类型)来进行跳转。使用跳转来实现两个网站的切换十分简单,需要的代码量也不多。尽管使用这种方法对于开发者来说可以降低复杂程度,但是它同样会降低用户体验。一个可替代的方法就是如果两者在同一台服务器上时使用 Alias 和 mod_rewrite 实现。如果是因为域名的不同而采用跳转的,那么可以通过使用 Alias 或者 mod_rewrite 建立 CNAME(保存一个域名和另外一个域名之间关系的 DNS 记录)来替代。

17.5.4 可缓存的 Ajax

Ajax 经常被提及的一个好处就是由于其从后台服务器传输信息的异步性而为用户带来的反馈的即时性。但是，使用 Ajax 并不能保证用户不会在等待异步的 JavaScript 和 XML 响应上花费时间。在很多应用中，用户是否需要等待响应取决于 Ajax 如何来使用。例如，在一个基于 Web 的 Email 客户端中，用户必须等待 Ajax 返回符合它们条件的邮件查询结果。记住一点，“异步”并不意味着“即时”，这很重要。

为了提高性能，优化 Ajax 响应是很重要的。提高 Ajax 性能的措施中最重要的方法就是使响应具有可缓存性，具体的讨论可以查看 Add an Expires or a Cache-Control Header，其他的几条规则也同样适用于 Ajax：

(1) Gizp 压缩文件；

(2) 减少 DNS 查找次数；

(3) 精简 JavaScript；

(4) 避免跳转；

(5) 配置 ETags。

来看一个例子：一个 Web 2.0 的 Email 客户端会使用 Ajax 来自动完成对用户地址簿的下载。如果用户在上次使用过 Email Web 应用程序后没有对地址簿做任何修改，而且 Ajax 响应通过 Expire 或者 Cache-Control 头来实现缓存，那么就可以直接从上一次的缓存中读取地址簿了。必须告知浏览器是使用缓存中的地址簿还是发送一个新的请求。这可以通过为读取地址簿的 Ajax URL 增加一个含有上次编辑时间的时间戳来实现，例如，&t=11900241612 等。如果地址簿在上次下载后没有被编辑过，时间戳就不变，则从浏览器的缓存中加载从而减少一次 HTTP 请求过程。如果用户修改过地址簿，时间戳就会用来确定新的 URL 和缓存响应并不匹配，浏览器就会重新请求更新地址簿。

即使 Ajax 响应是动态生成的，哪怕它只适用于一个用户，那么它也应该被缓存起来。这样做可以使 Web2.0 应用程序更加快捷。

17.5.5 推迟加载内容

可以仔细看一下自己的网页，问问自己“哪些内容是页面呈现时必需首先加载的？哪些内容和结构可以稍后再加载？”

把整个过程按照 onload 事件分隔成两部分，JavaScript 是一个理想的选择。例如，有用于实现拖放和动画的 JavaScript，那么它就应等待稍后加载，因为页面上的拖放元素是在初始化呈现之后才发生的。其他的例如隐藏部分的内容(用户操作之后才显现的内容)和处于折叠部分的图像也可以推迟加载。

工具可以节省工作量：YUI Image Loader 可以帮用户推迟加载折叠部分的图片，YUI Get utility 是包含 JS 和 CSS 的便捷方法。例如可以打开 Firebug 的 Net 选项卡看一下 Yahoo 的首页。

当性能目标和其他网站开发实践一致时就会相得益彰。这种情况下，通过程序提高网站性能的方法告诉人们，在支持JavaScript的情况下，可以先去除用户体验，不过这要保证网站在没有JavaScript时也可以正常运行。在确定页面运行正常后，再加载脚本来实现如拖放和动画等更加花哨的效果。

17.5.6 预加载

预加载和后加载看起来似乎恰恰相反，但实际上预加载是为了实现另外一种目标。预加载是在浏览器空闲时请求将来可能会用到的页面内容(如图像、样式表和脚本)。使用这种方法，当用户要访问下一个页面时，页面中的内容大部分已经加载到缓存中了，因此可以大大改善访问速度。

下面提供了几种预加载方法。

(1) 无条件加载：触发onload事件时，直接加载额外的页面内容。以Google.com为例，可以看一下它的spirit image图像是怎样在onload中加载的。这个spirit image图像在google.com主页中是不需要的，但是却可以在搜索结果页面中用到它。

(2) 有条件加载：根据用户的操作来有根据地判断用户下面可能去往的页面以及相应的预加载页面内容。在search.yahoo.com中可以看到如何在输入内容时加载额外的页面内容。

(3) 有预期的加载：载入重新设计过的页面时使用预加载。这种情况经常出现在页面经过重新设计后，用户抱怨"新的页面看起来很酷，但是却比以前慢"，问题可能出在用户对于旧站点建立了完整的缓存，而对于新站点却没有任何缓存内容。因此可以在访问新站之前就加载一部内容来避免这种结果的出现。在旧站中利用浏览器的空余时间加载新站中用到的图像和脚本来提高访问速度。

17.5.7 减少DOM元素数量

一个复杂的页面意味着需要下载更多数据，同时也意味着JavaScript遍历DOM的效率越慢。例如当增加一个事件句柄时，在500和5000个DOM元素中的循环效果肯定是不一样的。

大量DOM元素的存在意味着页面中有可以不用移除内容只需要替换元素标签就可以精简的部分。你在页面布局中使用表格了吗？你有没有仅仅为了布局而引入更多的<div>元素呢？也许会存在一个语意更适合或更贴切的标签可以供你使用。

YUI CSS utilities可以给布局带来巨大的帮助：grids.css可以帮用户实现整体布局，font.css和reset.css可以帮助用户移除浏览器默认格式。它提供了一个重新审视页面中的标签的机会，例如只有在语意上有意义时才使用<div>，而不是因为它具有换行效果才使用它。

DOM元素数量很容易计算出来，只需要在Firebug的控制台内输入：

```
document.getElementsByTagName('*').length
```

那么多少个 DOM 元素算是多呢？这可以对照有很好标记使用的类似页面。例如 Yahoo！主页是一个内容非常多的页面，但是它只使用了 700 个元素（HTML 标签）。

17.5.8 根据域名划分页面内容

把页面内容划分成若干部分，从而最大限度地实现平行下载。由于 DNS 查找带来的影响，首先要确保使用的域名数量为 2～4 个。例如，可以把用到的 HTML 内容和动态内容放在 www. example. org 上，而把页面各种组件（图片、脚本、CSS）分别存放在 statics1. example. org 和 statics. example. org 上。

可以在 Tenni Theurer 和 Patty Chi 合写的文章 Maximizing Parallel Downloads in the Carpool Lane 找到更多相关信息。

17.5.9 使 iframe 的数量最小

iframe 元素可以在父文档中插入一个新的 HTML 文档。了解 iframe 的工作原理后才能更加有效地使用它，这一点很重要。

1. iframe 优点

（1）解决加载缓慢的第三方内容如图标和广告等的加载问题；
（2）Security sandbox；
（3）并行加载脚本。

2. iframe 的缺点

（1）即时内容为空，加载也需要时间；
（2）会阻止页面加载；
（3）没有语意。

17.5.10 不要出现 404 错误

HTTP 请求时间消耗是很大的，因此使用 HTTP 请求来获得一个没有用处的响应（例如 404 没有找到页面）是完全没有必要的，它只会降低用户体验而不会有一点好处。

有些站点把 404 错误响应页面改为“你是不是要找***”，这虽然改进了用户体验但是同样也会浪费服务器资源（如数据库等）。最糟糕的情况是指向外部 JavaScript 的链接出现问题并返回 404 代码。首先，这种加载会破坏并行加载；其次浏览器会把试图在返回的 404 响应内容中找到可能有用的部分当作 JavaScript 代码来执行。

17.5.11 使用内容分发网络

用户与网站服务器的接近程度会影响响应时间的长短。把网站内容分散到多个、处于不同地域位置的服务器上可以加快下载速度，但是首先应该做些什么呢？

按地域布置网站内容的第一步并不是要尝试重新架构网站，让它们在分发服务器上正常运行。根据应用的需求来改变网站结构，这可能会包括一些比较复杂的任务，如在服务器间同步Session状态和合并数据库更新等。要想缩短用户和内容服务器的距离，这些架构步骤可能是不可避免的。

要记住，在终端用户的响应时间中有80%～90%的响应时间用于下载图像、样式表、脚本、Flash等页面内容，这就是网站性能黄金守则。和重新设计应用程序架构这样比较困难的任务相比，首先来分布静态内容会更好一点。这不仅会缩短响应时间，而且对于内容分发网络来说它更容易实现。

内容分发网络(Content Delivery Network，CDN)是由一系列分散到各个不同地理位置上的Web服务器组成的，它提高了网站内容的传输速度。用于向用户传输内容的服务器主要是根据和用户在网络上的靠近程度来指定的。例如，拥有最少网络跳数(Network Hops)和响应速度最快的服务器会被选定。

一些大型的网络公司拥有自己的CDN，但是使用像Akamai Technologies，Mirror Image Internet，或者Limelight Networks这样的CDN服务成本却非常高。对于刚刚起步的企业和个人网站来说，可能没有使用CDN的成本预算，但是随着目标用户群的不断扩大和更加全球化，CDN就是实现快速响应所必需的了。以Yahoo来说，它们转移到CDN上的网站程序静态内容节省了终端用户20%以上的响应时间。使用CDN是一个只需要相对简单地修改代码实现显著改善网站访问速度的方法。

17.5.12 为文件头指定Expires或Cache-Control

这条守则包括两方面的内容。

对于静态内容：设置文件头过期时间Expires的值为Never expire(永不过期)。

对于动态内容：使用恰当的Cache-Control文件头来帮助浏览器进行有条件的请求。

网页内容设计现在越来越丰富，这就意味着页面中要包含更多的脚本、样式表、图片和Flash。第一次访问页面的用户就意味着进行多次的HTTP请求，但是通过使用Expires文件头就可以使这样的内容具有缓存性。它避免了接下来的页面访问中不必要的HTTP请求。Expires文件头经常用于图像文件，但是应该在所有的内容中都使用它，包括脚本、样式表和Flash等。

浏览器(和代理)使用缓存来减少HTTP请求的大小和次数以加快页面访问速度。Web服务器在HTTP响应中使用Expires文件头来告诉客户端内容需要缓存多长时间。下面这个例子是一个较长时间的Expires文件头，它告诉浏览器这个响应直到2010年4月15日才过期。

```
Expires: Thu, 15 Apr 2010 20:00:00 GMT
```

如果使用的是 Apache 服务器，可以使用 ExpiresDefault 来设定相对当前日期的过期时间。下面这个例子是使用 ExpiresDefault 来设定请求时间后 10 年过期的文件头：

```
ExpiresDefault "access plus 10 years"
```

要切记，如果使用了 Expires 文件头，当页面内容改变时就必须改变内容的文件名。按 Yahoo! 来说人们经常使用这样的步骤：在内容的文件名中加上版本号，如 yahoo_2.0.6.js。

使用 Expires 文件头只会在用户已经访问过该网站后才会起作用。当用户首次访问网站时，这对减少 HTTP 请求次数来说是无效的，因为浏览器的缓存是空的。因此这种方法对于网站性能的改进情况要依据它们“预缓存”存在时对页面的单击频率（“预缓存”中已经包含了页面中的所有内容）。Yahoo! 建立了一套测量方法，人们发现所有的页面浏览量中有 75%～85%都有“预缓存”。通过使用 Expires 文件头，增加了缓存在浏览器中内容的数量，并且可以在用户接下来的请求中再次使用这些内容，这甚至都不需要通过用户发送一个字节的请求。

17.5.13 Gzip 压缩文件内容

网络传输中的 HTTP 请求和应答时间可以通过前端机制得到显著改善。的确，终端用户的带宽、互联网提供者、与对等交换点的靠近程度等都不是网站开发者所能决定的，但是还有其他因素影响着响应时间，通过减小 HTTP 响应的大小可以节省 HTTP 响应时间。

从 HTTP/1.1 开始，Web 客户端都默认支持 HTTP 请求中有 Accept-Encoding 文件头的压缩格式：

```
Accept-Encoding: gzip, deflate
```

如果 Web 服务器在请求的文件头中检测到上面的代码，就会以客户端列出的方式压缩响应内容。Web 服务器把压缩方式通过响应文件头中的 Content-Encoding 来返回给浏览器。

```
Content-Encoding: gzip
```

Gzip 是目前最流行也是最有效的压缩方式。这是由 GNU 项目开发并通过 RFC 1952 来标准化的。另外仅有的一个压缩格式是 deflate，但是它的使用范围有限，效果也稍稍逊色。

Gzip 大概可以减少 70%的响应规模。目前大约有 90%通过浏览器传输的互联网交换支持 Gzip 格式。如果使用的是 Apache，Gzip 模块配置和版本有关：Apache 1.3 使用 mod_zip，而 Apache 2.x 使用 moflate。

浏览器和代理都会存在这样的问题：浏览器期望收到的和实际接收到的内容会存在不匹配的现象。幸好，这种特殊情况随着旧式浏览器使用量的减少在减少。Apache 模块

会通过自动添加适当的 Vary 响应文件头来避免这种状况的出现。

服务器根据文件类型来选择需要进行 gzip 压缩的文件，但是这过于限制了可压缩的文件。大多数 Web 服务器会压缩 HTML 文档。对脚本和样式表进行压缩同样也是值得做的事情，但是很多 Web 服务器都没有这个功能。实际上，压缩任何一个文本类型的响应，包括 XML 和 JSON，都是值得的。图像和 PDF 文件由于已经压缩过了所以不能再进行 Gzip 压缩。如果试图 Gizp 压缩这些文件的话，不但会浪费 CPU 资源还会增加文件的大小。

Gzip 压缩所有可能的文件类型是减少文件体积增加用户体验的简单方法。

17.5.14 配置 ETag

Entity Tag(ETag)(实体标签)是 Web 服务器和浏览器用于判断浏览器缓存中的内容和服务器中的原始内容是否匹配的一种机制（“实体”就是所说的“内容”，包括图片、脚本、样式表等）。增加 ETag 为实体的验证提供了一个比使用 last-modified date(上次编辑时间)更加灵活的机制。ETag 是一个识别内容版本号的唯一字符串。唯一的格式限制就是它必须包含在双引号内。原始服务器通过含有 ETag 文件头的响应指定页面内容的 ETag。

```
HTTP/1.1 200 OK
Last-Modified: Tue, 12 Dec 2006 03:03:59 GMT
ETag: "10c24bc-4ab-457e1c1f"
Content-Length: 12195
```

稍后，如果浏览器要验证一个文件，它会使用 If-None-Match 文件头来把 ETag 传回给原始服务器。在这个例子中，如果 ETag 匹配，就会返回一个 304 状态码，这就节省了 12195 字节的响应。

```
GET /i/yahoo.gif HTTP/1.1
Host: us.yimg.com
If-Modified-Since: Tue, 12 Dec 2006 03:03:59 GMT
If-None-Match: "10c24bc-4ab-457e1c1f"
HTTP/1.1 304 Not Modified
```

ETag 的问题在于，它是根据可以辨别网站所在的服务器的具有唯一性的属性来生成的。当浏览器从一台服务器上获得页面内容后到另外一台服务器上进行验证时 ETag 就会不匹配，这种情况对于使用服务器组和处理请求的网站来说是非常常见的。默认情况下，Apache 和 IIS 都会把数据嵌入 ETag 中，这会显著减少多服务器间的文件验证冲突。

Apache 1.3 和 2.x 中的 ETag 格式为 inode-size-timestamp。即使某个文件在不同的服务器上会处于相同的目录下，文件大小、权限、时间戳等都完全相同，但是在不同服务器上它们的内码也是不同的。

IIS 5.0 和 IIS 6.0 处理 ETag 的机制相似。IIS 中的 ETag 格式为 Filetimestamp：ChangeNumber。用 ChangeNumber 来跟踪 IIS 配置的改变。网站所用的不同 IIS 服务器间的 ChangeNumber 也不相同。不同的服务器上的 Apache 和 IIS 即使对于完全相同的内容产生的 ETag 在也不相同，用户并不会接收到一个小而快的 304 响应；相反它们会接收一个正常的 200 响应并下载全部内容。如果网站只放在一台服务器上，就不会存在这个问题。但是如果网站是架设在多个服务器上，并且使用 Apache 和 IIS 产生默认的 ETag 配置的，用户获得页面就会相对慢一点，服务器会传输更多的内容，占用更多的带宽，代理也不会有效地缓存网站内容。即使内容拥有 Expires 文件头，无论用户什么时候单击“刷新”或者“重载”按钮都会发送相应的 GET 请求。

如果没有使用 ETag 提供的灵活的验证模式，那么干脆把所有的 ETag 都去掉会更好。Last-Modified 文件头验证是基于内容的时间戳的。去掉 ETag 文件头会减少响应和下次请求中文件的大小。微软的这篇支持文稿讲述了如何去掉 ETag。在 Apache 中，只需要在配置文件中简单添加下面一行代码就可以了：

```
FileETag none
```

17.5.15 尽早刷新输出缓冲

当用户请求一个页面时，无论如何都会花费 200～500ms 用于后台组织 HTML 文件。在这期间，浏览器会一直空闲等待数据返回。在 PHP 中，可以使用 flush()方法，它允许用户把已经编译好的部分 HTML 响应文件先发送给浏览器，这时浏览器就可以下载文件中的内容(脚本等)而后台同时处理剩余的 HTML 页面。这样做的效果会在后台繁忙或者前台较空闲时更加明显。

输出缓冲应用最好的一个地方就是紧跟在<head />之后，因为 HTML 的头部分容易生成而且头部往往包含 CSS 和 JavaScript 文件，这样浏览器就可以在后台编译剩余 HTML 的同时并行下载它们，例如：

```
...<!--css, js-->
</head>
<?php flush(); ?>
<body>
...<!--content-->
```

为了证明使用这项技术的好处，Yahoo! 搜索率先研究并完成了用户测试。

17.5.16 使用 GET 来完成 Ajax 请求

Yahoo! Mail 团队发现，当使用 XMLHttpRequest 时，浏览器中的 POST 方法是一个“两步走”的过程：首先发送文件头，然后才发送数据。因此使用 GET 最为恰当，因为它只需发送一个 TCP 包(除非有很多 cookie)。IE 中 URL 的最大长度为 2K，因此如果

要发送一个超过 2K 的数据时就不能使用 GET 了。

一个有趣的不同就是 POST 并不像 GET 那样实际发送数据。根据 HTTP 规范，GET 意味着“获取”数据，因此当仅仅获取数据时使用 GET 更加有意义(从语意上讲也是如此)，相反，发送并在服务端保存数据时使用 POST。

17.5.17 把样式表置于顶部

在研究 Yahoo! 的性能表现时，笔者发现把样式表放到文档的<head />内部似乎会加快页面的下载速度。这是因为把样式表放到<head />内会使页面有步骤地加载显示。

注重性能的前端服务器往往希望页面有秩序地加载。同时，也希望浏览器把已经接收到的内容尽可能显示出来。这对于拥有较多内容的页面和网速较慢的用户来说特别重要。向用户返回可视化的反馈，如进程指针，已经有了较好的研究并形成了正式文档。在研究中，HTML 页面就是进程指针。当浏览器有序地加载文件头、导航栏、顶部的 logo 等时，对于等待页面加载的用户来说都可以作为可视化的反馈，这从整体上改善了用户体验。把样式表放在文档底部的问题是在包括 Internet Explorer 在内的很多浏览器中这会中止内容的有序呈现。浏览器中止呈现是为了避免样式改变引起的页面元素重绘。用户不得不面对一个空白页面。

HTML 规范清楚地指出样式表要放在包含在页面的<head />区域内。和<a />不同，<link />只能出现在文档的<head />区域内，尽管可以多次使用它。无论是引起白屏还是出现没有样式化的内容都不值得去尝试。最好的方案就是按照 HTML 规范在文档<head />内加载样式表。

17.5.18 避免使用 CSS 表达式

CSS 表达式(Expression)是动态设置 CSS 属性的强大(但危险)方法。Internet Explorer 从第 5 个版本开始支持 CSS 表达式。下面的例子中，使用 CSS 表达式可以实现隔一个小时切换一次背景颜色：

```
background-color: expression( (new Date()).getHours()%2 ? "#B8D4FF" :
"#F08A00" );
```

如上所示，Expression 中使用了 JavaScript 表达式。CSS 属性根据 JavaScript 表达式的计算结果来设置。Expression 方法在其他浏览器中不起作用，因此在跨浏览器的设计中单独针对 Internet Explorer 设置时会比较有用。

表达式的问题就在于它的计算频率要比人们想象得多。不仅仅是在页面显示和缩放时，就是在页面滚动乃至移动鼠标时都要重新计算一次。给 CSS 表达式增加一个计数器可以跟踪表达式的计算频率。在页面中随便移动鼠标都可以轻松达到 10 000 次以上的计算量。

一个减少 CSS 表达式计算次数的方法就是使用一次性的表达式，它在第一次运行时将结果赋给指定的样式属性，并用这个属性来代替 CSS 表达式。如果样式属性必须在页面周期内动态地改变，使用事件句柄来代替 CSS 表达式是一个可行的办法。如果必须使用 CSS 表达式，一定要记住它们要计算成千上万次并且可能会对页面的性能产生影响。

17.5.19 使用外部 JavaScript 和 CSS

很多性能规则都是关于如何处理外部文件的。但是，在采取这些措施前可能会问一个更基本的问题：JavaScript 和 CSS 是应该放在外部文件中还是把它们放在页面本身内呢？

在实际应用中使用外部文件可以提高页面速度，因为 JavaScript 和 CSS 文件都能在浏览器中产生缓存。内置在 HTML 文档中的 JavaScript 和 CSS 则会在每次请求中随 HTML 文档重新下载，这虽然减少了 HTTP 请求的次数，却增加了 HTML 文档的大小。从另一方面来说，如果外部文件中的 JavaScript 和 CSS 被浏览器缓存，在没有增加 HTTP 请求次数的同时可以减少 HTML 文档的大小。

关键问题是，外部 JavaScript 和 CSS 文件缓存的频率和请求 HTML 文档的次数有关。虽然有一定的难度，但是仍然有一些指标可以测量它。如果一个会话中用户会浏览网站中的多个页面，并且这些页面中会重复使用相同的脚本和样式表，缓存外部文件就会带来更大的益处。

许多网站没有功能建立这些指标。对于这些网站来说，最好的解决方法就是把 JavaScript 和 CSS 作为外部文件引用。比较适合使用内置代码的例外就是网站的主页，如 Yahoo！主页和 My Yahoo!。主页在一次会话中拥有较少(可能只有一次)的浏览量，可以发现内置 JavaScript 和 CSS 对于终端用户来说会加快响应时间。

对于拥有较大浏览量的首页来说，有一种技术可以平衡内置代码带来的 HTTP 请求减少与通过使用外部文件进行缓存带来的好处。其中一个就是在首页中内置 JavaScript 和 CSS，但是在页面下载完成后动态下载外部文件，在子页面中使用到这些文件时，它们已经缓存到浏览器了。

17.5.20 削减 JavaScript 和 CSS

精简是指从去除代码不必要的字符减少文件大小从而节省下载时间。消减代码时，所有的注释、不需要的空白字符(空格、换行、Tab 缩进)等都要去掉。在 JavaScript 中，由于需要下载的文件体积变小了，从而节省了响应时间。精简 JavaScript 中目前用到的最广泛的两个工具是 JSMin 和 YUI Compressor。YUI Compressor 还可用于精简 CSS。

混淆是另外一种可用于源代码优化的方法。这种方法要比精简复杂一些并且在混淆的过程更易产生问题。在对美国前 10 大网站的调查中发现，精简也可以缩小原来代码体积的 21%，而混淆可以达到 25%。尽管混淆法可以更好地缩减代码，但是对于 JavaScript 来说精简的风险更小。

除消减外部的脚本和样式表文件外，<script>和<style>代码块也可以并且应该进行消减。即使用 Gzip 压缩过脚本和样式表，精简这些文件仍然可以节省 5%以上的空间。由于 JavaScript 和 CSS 的功能和体积的增加，消减代码将会获得益处。

17.5.21　用<link>代替@import

前面的最佳实现中提到 CSS 应该放置在顶端以利于有序加载呈现。

在 IE 中，页面底部@import 和使用<link>的作用是一样的，因此最好不要使用它。

17.5.22　避免使用滤镜

IE 独有的属性 AlphaImageLoader 用于修正 7.0 以下版本中显示 PNG 图片的半透明效果。这个滤镜的问题在于浏览器加载图片时会终止内容的呈现并且冻结浏览器。在每一个元素（不仅仅是图片）它都会运算一次，增加了内存开支，因此它的问题是多方面的。

完全避免使用 AlphaImageLoader 的最好方法就是使用 PNG8 格式来代替，这种格式能在 IE 中很好地工作。如果确实需要使用 AlphaImageLoader，请使用下划线_filter 又使之对 IE7 以上版本的用户无效。

17.5.23　把脚本置于页面底部

脚本带来的问题就是它阻止了页面的平行下载。HTTP/1.1 规范建议，浏览器每个主机名的并行下载内容不超过两个。如果图片放在多个主机名上，可以在每个并行下载中同时下载两个以上的文件。但是当下载脚本时，浏览器就不会同时下载其他文件了，即便主机名不相同。

在某些情况下把脚本移到页面底部可能不太容易。例如，如果脚本中使用了 document.write 来插入页面内容，它就不能被往下移动了，这里可能还会有作用域的问题。很多情况下，都会遇到这方面的问题。

一个经常用到的替代方法就是使用延迟脚本。DEFER 属性表明脚本中没有包含 document.write，它告诉浏览器继续显示。不幸的是，Firefox 并不支持 DEFER 属性。在 Internet Explorer 中，脚本可能会被延迟但效果也不会像人们所期望的那样。如果脚本可以被延迟，那么它就可以移到页面的底部，这会让页面加载得快一点。

17.5.24　剔除重复脚本

在同一个页面中重复引用 JavaScript 文件会影响页面的性能，人们可能会认为这种情况并不多见。对于美国前 10 大网站的调查显示其中有两家存在重复引用脚本的情况。有两种主要因素导致一个脚本被重复引用的奇怪现象发生：团队规模和脚本数量。如果

真的存在这种情况，重复脚本会引起不必要的 HTTP 请求和无用的 JavaScript 运算，这降低了网站性能。

在 Internet Explorer 中会产生不必要的 HTTP 请求，而在 Firefox 却不会。在 Internet Explorer 中，如果一个脚本被引用两次而且又不可缓存，它就会在页面加载过程中产生两次 HTTP 请求。即时脚本可以缓存，当用户重载页面时也会产生额外的 HTTP 请求。

除增加额外的 HTTP 请求外，多次运算脚本也会浪费时间。在 Internet Explorer 和 Firefox 中不管脚本是否可缓存，它们都存在重复运算 JavaScript 的问题。

一个避免偶尔发生的两次引用同一脚本的方法是在模板中使用脚本管理模块引用脚本。在 HTML 页面中使用<script />标签引用脚本的最常见的方法就是：

```
<script type="text/javascript" src="menu_1.0.17.js"></script>
```

在 PHP 中可以通过创建名为 insertScript 的方法来替代：

```
<?php insertScript("menu.js") ?>
```

为了防止多次重复引用脚本，这个方法中还应该使用其他机制来处理脚本，如检查所属目录和在脚本文件名中增加版本号以用于 Expire 文件头等。

17.5.25　减少 DOM 访问

使用 JavaScript 访问 DOM 元素比较慢，因此为了获得更多的应该页面，应该做到：

(1) 缓存已经访问过的有关元素；

(2) 线下更新完节点之后再将它们添加到文档树中；

(3) 避免使用 JavaScript 来修改页面布局。

有关此方面的更多信息请查看 Julien Lecomte 在 YUI 专题中的文章“高性能 Ajax 应用程序”。

17.5.26　开发智能事件处理程序

有时候人们会感觉到页面反应迟钝，这是因为 DOM 树元素中附加了过多的事件句柄并且一些事件句病被频繁地触发。这就是为什么说使用 event delegation(事件代理)是一种好方法了。如果在一个 div 中有 10 个按钮，用户只需要在 div 上附加一次事件句柄就可以了，而不用去为每一个按钮增加一个句柄。事件冒泡时可以捕捉到事件并判断出是哪个事件发出的。

同样也不用为了操作 DOM 树而等待 onload 事件的发生。用户需要做的就是等待树结构中要访问的元素出现，也不用等待所有图像都加载完毕。

用户可能会希望用 DOMContentLoaded 事件来代替 onload，但是在所有浏览器都支持它之前可使用 YUI 事件应用程序中的 onAvailable 方法。

17.5.27 减小 cookie 体积

HTTP cookie 可以用于权限验证和个性化身份等多种用途。cookie 内的有关信息是通过 HTTP 文件头在 Web 服务器和浏览器之间进行交流的。因此保持 cookie 尽可能小以减少用户的响应时间十分重要。

更多有关信息可以查看 Tenni Theurer 和 Patty Chi 的文章 *When the Cookie Crumbles*。这些研究中主要包括：

（1）去除不必要的 cookie；

（2）使 cookie 体积尽量小以减少对用户响应的影响；

（3）注意在适应级别的域名上设置 cookie 以便使子域名不受影响；

（4）设置合理的过期时间。较早的 Expire 时间和不要过早地清除 cookie，都会改善用户的响应时间。

17.5.28 对于页面内容使用无 cookie 域名

当浏览器在请求中同时请求一张静态的图片和发送 cookie 时，服务器对于这些 cookie 不会做任何使用。因此它们只是因为某些负面因素而创建的网络传输。所有应该确定对于静态内容的请求是无 cookie 的请求。创建一个子域名并用它来存放所有静态内容。

如果域名是 www. example. org，可以在 static. example. org 上存在静态内容。但是，如果不是在 www. example. org 上而是在顶级域名 example. org 设置了 cookie，那么所有对于 static. example. org 的请求都包含 cookie。在这种情况下，可以再重新购买一个新的域名来存放静态内容，并且要保持这个域名是无 cookie 的。Yahoo！使用的是 ymig. com，YouTube 使用的是 ytimg. com，Amazon 使用的是 images-anazon. com 等。

使用无 cookie 域名存放静态内容的另外一个好处就是一些代理（服务器）可能会拒绝对 cookie 的内容请求进行缓存。一个相关的建议就是，如果想确定应该使用 example . org 还是 www. example. org 作为一个主页，要考虑到 cookie 带来的影响。忽略掉 www 会使用户除了把 cookie 设置到 *. example. org（* 是泛域名解析，代表了所有子域名译者 dudo 注）外没有其他选择，因此出于性能方面的考虑最好使用带有 www 的子域名并且在它上面设置 cookie。

17.5.29 优化图像

设计人员完成对页面的设计之后，不要急于将它们上传到 Web 服务器，这里还需要做几件事：

检查一下 GIF 图片中图像颜色的数量是否和调色板规格一致。使用 imagemagick 中以下的命令行很容易检查：

```
identify -verbose image.gif
```

如果发现图片中只用到了 4 种颜色，而在调色板中显示了 256 色的颜色槽，那么这张图片就还有压缩的空间。

尝试把 GIF 格式转换成 PNG 格式，看看是否节省空间。大多数情况下是可以压缩的。由于浏览器支持有限，设计者们往往不太乐意使用 PNG 格式的图片，不过这都是过去的事情了。现在只有一个问题就是在真彩 PNG 格式中的 alpha 通道半透明问题，不过同样地，GIF 也不是真彩格式也不支持半透明。因此 GIF 能做到的，PNG(PNG8)同样也能做到(除了动画)。下面这条简单的命令可以安全地把 GIF 格式转换为 PNG 格式：

```
convert image.gif image.png
```

这里要说的是："给 PNG 一个施展身手的机会吧！"

在所有的 PNG 图片上运行 pngcrush(或者其他 PNG 优化工具)。例如：

```
pngcrush image.png -rem alla -reduce -brute result.png
```

在所有的 JPEG 图片上运行 jpegtran。这个工具可以对图片中出现的锯齿等做无损操作，同时它还可以用于优化和清除图片中的注释以及其他无用信息(如 EXIF 信息)：

```
jpegtran -copy none -optimize -perfect src.jpg dest.jpg
```

17.5.30 优化 CSS Spirite

在 Spirite 中水平排列图片，垂直排列会稍稍增加文件大小；在 Spirite 中把颜色较近的组合在一起可以降低颜色数，理想状况是低于 256 色以便适用 PNG8 格式；为了便于移动，不要在 Spirite 的图像中间留有较大空隙。这虽然不大会增加文件大小但对于用户代理来说它需要更少的内存来把图片解压为像素地图。

17.5.31 不要在 HTML 中缩放图像

不要为了在 HTML 中设置长宽而使用比实际需要大的图片。如果需要：

```
<img width="100" height="100" src="mycat.jpg" alt="My Cat" />
```

那么图片(mycat.jpg)就应该是 100×100 像素而不是把一个 500×500 像素的图片缩小使用。

17.5.32 favicon.ico 要小而且可缓存

favicon.ico 是位于服务器根目录下的一个图片文件。它是必定存在的，因为即使不关心它是否有用，浏览器也会对它发出请求，因此最好不要返回一个 404 Not Found 的响应。由于是在同一台服务器上的，它每被请求一次 cookie 就会被发送一次。这个图片文

件还会影响下载顺序,例如在 IE 中,当在 onload 中请求额外的文件时,favicon 会在这些额外内容被加载前下载。

因此,为了减少 favicon.ico 带来的弊端,要做到:

(1) 文件尽量地小,最好小于 1KB;

(2) 在适当的时候(也就是不打算再换 favicon.ico 的时候,因为更换新文件时不能对它进行重命名)为它设置 Expires 文件头。可以很安全地把 Expires 文件头设置为未来的几个月,可以通过核对当前 favicon.ico 上次的编辑时间来做出判断。

Imagemagick 可以帮用户创建小巧的 favicon。

17.5.33 保持单个内容小于 25KB

这条限制主要是因为 iPhone 不能缓存大于 25KB 的文件,注意这里指的是解压缩后的大小。由于单纯 Gzip 压缩可能达不要求,因此精简文件就显得十分重要。

查看更多信息,请参阅 Wayne Shea 和 Tenni Theurer 的文章 *Performance Research*, *Part 5*: *iPhone Cacheability —Making it Stick*。

17.5.34 打包组件成复合文本

把页面内容打包成复合文本就如同带有多附件的 Email,它能够使用户在一个 HTTP 请求中取得多个组件(切记: HTTP 请求是很奢侈的)。当使用这条规则时,首先要确定用户代理是否支持(iPhone 就不支持)。

17.6 本章小结

本章详细介绍了 Web 前端相关的基础知识、Web 前端性能的重要性以及如何使用工具评估 Web 前端性能。随着客户端技术的发展,Web 前端性能已经成为 Web 应用性能的一个主要研究方向。

参考文献

[1] 段念.软件性能测试过程详解与案例剖析(第二版).北京:清华大学出版社,2012.
[2] 陈能技,郭柏雅.性能测试诊断分析与优化.北京:电子工业出版社,2012.
[3] 于涌,王磊,曹向志,等.精通软件性能测试与LoadRunner最佳实战.北京:人民邮电出版社,2013.
[4] Patton.软件测试(第二版).张小松,等,译.北京:机械工业出版社,2006.
[5] 施迎.Web性能测试实战详解.北京:清华大学出版社,2013.
[6] 魏娜娣,李文斌,裴军霞.软件性能测试——基于loadrunner应用.北京:清华大学出版社,2012.
[7] 赵斌.软件测试技术经典教程(第二版).北京:科学出版社,2011.
[8] 黄文高.LoadRunner性能测试完全讲义(第二版).北京:水利水电出版社,2014.
[9] 柳纯录,黄子河,陈渌萍.软件评测师教程.北京:清华大学出版社,2005.
[10] 陈绍英.Web性能测试实战.北京:电子工业出版社,2006.
[11] Mathur.软件测试基础教程.王峰,郭长国,陈振华,等,译.北京:人民邮电出版社,2011.
[12] 陈霁.性能测试进阶指南:LoadRunner 11实战.北京:电子工业出版社,2012.
[13] 朱少民.软件测试方法和技术(第二版).北京:清华大学出版社,2010.
[14] 柳胜.性能测试从零开始:LoadRunner入门.北京:电子工业出版社,2008.
[15] 于涌.精通软件性能测试与LoadRunner实战.北京:人民邮电出版社,2010.
[16] Molyneaux.应用程序性能测试的艺术.李刚,等,译.北京:机械工业出版社,2010.
[17] 陈绍英,夏海涛,金成姬.Web性能测试实战.北京:电子工业出版社,2006.
[18] 于涌.精通软件性能测试与LoadRunner实战.北京:人民邮电出版社,2010.
[19] 陈能技.软件测试技术大全:测试基础、流行工具与项目实战(第二版).北京:人民邮电出版社,2011.